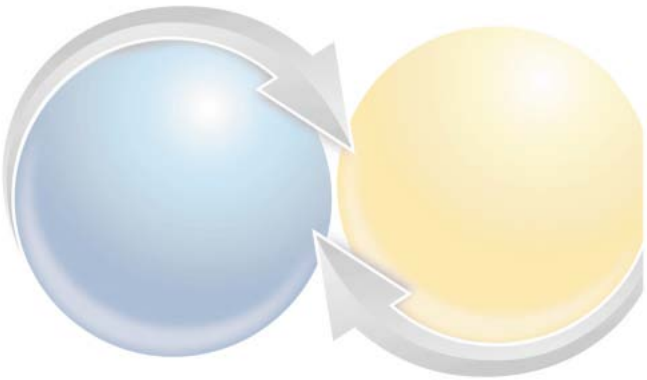
Livelink WCM Server

WCM WebServices Programmer's Manual

This manual describes the Web Services programming interface which allows you to access the functions and methods of Livelink WCM Server via the internet. Highlighted topics include:

- basic concepts of WCM WebServices
- using WCM WebServices
- access to administration data, authentication, and object management via WCM WebServices
- application examples



OPEN TEXT CORPORATION

Great Minds Working Together™

Copyright 2005 by Open Text Corporation.

The copyright to these materials and any accompanying software is owned, without reservation, by Open Text. These materials and any accompanying software may not be copied in whole or part without the express, written permission of Open Text.

Open Text Corporation is the owner of the trademarks Open Text, 'Great Minds Working Together', Livelink, and MeetingZone among others. This list is not exhaustive. All other products or company names are used for identification purposes only, and are trademarks of their respective owners. All rights reserved.

Open Text Corporation provides certain warranties and limitations in connection with the software that this document describes. For information about these warranties and limitations, refer to the license agreement entered into between the licensee and Open Text Corporation.

Contacting Us

Corporate Headquarters
Open Text Corporation
185 Columbia Street West,
Waterloo, Ontario
N2L 5Z5
Canada

(519) 888-7111

If you subscribe to our Software Maintenance Program or would like more information about additional support programs, visit Open Text Customer Support at http://www.opentext.com/services/support.html.

If you have suggestions for this publication, send an e-mail message to documentation@opentext.com to contact the Open Text Publications Group.

Visit our home page at http://www.opentext.com for more information about Open Text products and services.

© 2005 IXOS SOFTWARE AG

Bretonischer Ring 12
85630 Grasbrunn, Germany

Tel.: +49 (89) 4629-0
Fax: +49 (89) 4629-1199
eMail: <office@ixos.de>
Internet: http://www.ixos.de

All rights reserved, including those regarding reproduction, copying or other use or communication of the contents of this document or parts thereof. No part of this publication may be reproduced, transmitted to third parties, processed using electronic retrieval systems, copied, distributed or used for public demonstration in any form without the written consent of IXOS SOFTWARE AG. We reserve the right to update or modify the contents. Any and all information that appears within illustrations of screenshots is provided coincidentally to better demonstrate the functioning of the software. IXOS

SOFTWARE AG hereby declares that this information reflects no statistics of nor has any validity for any existing company. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) and software developed by the Apache Software Foundation (http://www.apache.org/).

Trademarks IXOS: IXOS SOFTWARE AG.

SAP®, R/3® and SAP ArchiveLink® are registered trademarks of SAP AG.

Microsoft®, Microsoft Windows NT® and the names of further Microsoft products are registered trademarks of Microsoft Corporation.

Acrobat Reader Copyright © 1987 Adobe Systems Incorporated. All rights reserved. Adobe and Acrobat are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions.

Siebel® is a registered trademark by Siebel Systems, Inc.

Other product names are used only to identify the products and they may be registered trademarks of the relevant manufacturers.

Copyright © 2005 Gauss Interprise AG Hamburg, Gauss Interprise, Inc. Irvine, California. All rights reserved worldwide.

This document and the related software are property of Gauss Interprise AG or its suppliers and protected by copyright and other laws. They are distributed under licenses restricting their use, copying, distribution, and decompilation. Neither receipt nor possession of this document confers or transfers any right to reproduce or disclose any part of the contents hereof. No part of this document may be reproduced in any form by any means without prior written authorization of Gauss Interprise AG or Gauss Interprise, Inc.

Moreover, the regulations of the software license agreement apply to this documentation.

All brand names and trademarks mentioned are the property of their respective owners.

http://www.gaussvip.com

Program version: Livelink Web Content Management Server™ (Content Server) 9.5.0

Document version: En-02

Publication date: March 2005

Table of Contents

List of Figures 6

List of Tables 7

Chapter 1 Introduction 11

About this Document 12

Typographic Conventions 16

Chapter 2 Concepts 19

Web Services 19

WCM WebServices 25

Concepts of Livelink WCM Server 28

Chapter 3 Using WCM WebServices 37

Configuration 37

The WCM WebServices Description 42

Error Handling 45

Chapter 4 Access to Administration Data and Authentication 51

Users, Groups, and Roles 52

Functional Areas 59

Websites 61

Deployment Systems 63

Run Levels 64

General Requests 66

Authentication 67

Chapter 5 **Object Management** **71**
WCM-Specific Data Types 72
General Parameters 93
Staging Methods 95
Workflow Methods 104
Object Management Methods 114
Searching for WCM Objects 141
Chapter 6 **Application Examples** **149**
VisualBasic for Applications 150
C# and ASP.NET 170
Glossary **189**
Index **195**

List of Figures

Fig. 1 –	Roles and actions in the Web Service architecture	21
Fig. 2 –	Protocol stack for Web Services	23
Fig. 3 –	Method call of a Web Service client	25
Fig. 4 –	Basic principle of WCM WebServices	27
Fig. 5 –	Applications of the Content server	40
Fig. 6 –	Parameters of the application `WebServiceApplication`	41
Fig. 7 –	The data type `Filter` and derived special filter types	142
Fig. 8 –	Integrating the libraries in the VisualBasic Editor	156
Fig. 9 –	Creating a C# project	170
Fig. 10 –	Adding the WSDL as a web reference	171
Fig. 11 –	Entering user ID and password	172
Fig. 12 –	Automatically created classes in the namespace `VipWebServiceClient`	173
Fig. 13 –	The sample application (list of all PDF files)	174
Fig. 14 –	E-mail dialog of the example application	175

List of Tables

Table 1 –	Available functions of the Livelink WCM Server APIs	14
Table 2 –	Exceptions and possible causes	47
Table 3 –	Data types for users	52
Table 4 –	Data types for groups and roles	54
Table 5 –	Parameters of the method `getGroupProfile`	55
Table 6 –	Parameters of the method `getRoleProfile`	56
Table 7 –	Parameters of the method `getUserProfile`	57
Table 8 –	Names of the default functional areas	59
Table 9 –	Parameter of the method `getDeploymentSystems`	63
Table 10 –	Parameter of the method `changePassword`	68
Table 11 –	Parameter of the method `substituteLogin`	69
Table 12 –	The components of the data type `ObjectId`	73
Table 13 –	The components of the data type `ObjectData`	75
Table 14 –	The components of the data type `ObjectType`	80
Table 15 –	The individual object statuses	82
Table 16 –	The possible transitions between the object statuses	82
Table 17 –	The components of the data type `ObjectState`	84
Table 18 –	Possible access rights for WCM objects	85
Table 19 –	The components of the data type `Permission`	85
Table 20 –	The components of the data type `Acl`	86
Table 21 –	The components of the data type `AclEntry`	87
Table 22 –	The components of the data type `MultiImportPart`	88
Table 23 –	The components of the data type `Workflow`	90
Table 24 –	The components of the data type `Activity`	91
Table 25 –	The components of the data type `Transition`	92
Table 26 –	The components of the data type `WorkflowNavigationInfo`	92

Table 27 – The components of the data type DeploymentWaitInfo 93
Table 28 – The components of the data type EMailInfo 94
Table 29 – Parameters of the method checkIn 95
Table 30 – Parameters of the method checkOut 97
Table 31 – Parameters of the method directRelease 98
Table 32 – Parameters of the method reject 99
Table 33 – Parameters of the method release 100
Table 34 – Parameters of the method submit 101
Table 35 – Parameters of the method submitImmediately 102
Table 36 – Parameters of the method undoCheckOut 103
Table 37 – Parameters of the method assignWorkflow 104
Table 38 – Parameters of the method assignWorkflowAsync 106
Table 39 – Parameters of the method forward 107
Table 40 – Parameters of the method getAssignedJobs 108
Table 41 – Parameters of the method getAssignedWorkflow 108
Table 42 – Parameters of the method getWorkflow 110
Table 43 – Parameter of the method isForwardable 111
Table 44 – Parameter of the method isRemovable 111
Table 45 – Parameters of the method removeWorkflow 112
Table 46 – Parameters of the method removeWorkflowAsync 113
Table 47 – Parameters of the method addRemark 116
Table 48 – Parameters of the method change 117
Table 49 – Parameter of the method checkReferencesForDelete 118
Table 50 – Parameter of the method checkReferencesForRelease 119
Table 51 – Parameter of the method checkReferencesForSubmit 120
Table 52 – Parameters of the method convertContent 120
Table 53 – Parameters of the method copy 121
Table 54 – Parameters of the method create 123
Table 55 – Parameters of the method delete 124

Table 56 – Parameters of the method `depublishPage` 125
Table 57 – Parameters of the method `destroy` 126
Table 58 – Parameters of the method `generatePage` 127
Table 59 – Parameters of the method `get` 128
Table 60 – Parameters of the method `getCheckOutContent` 129
Table 61 – Parameters of the method `getChildren` 130
Table 62 – Parameters of the method `getContent` 131
Table 63 – Parameters of the method `getDeploymentJobs` 132
Table 64 – Parameter of the method `getExternalLinks` 133
Table 65 – Parameters of the method `getLastLogEntries` 133
Table 66 – Parameters of the method `getParent` 134
Table 67 – Parameter of the method `getVersionList` 135
Table 68 – Parameters of the method `move` 136
Table 69 – Parameters of the method `multiImport` 138
Table 70 – Parameters of the method `restoreVersion` 139
Table 71 – Parameters of the method `SortParentsFirst` 140
Table 72 – Attribute types and filters 143
Table 73 – Parameters of the method `filter` 146
Table 74 – Class modules 157
Table 75 – ASP.NET example 176

CHAPTER 1
Introduction

WCM WebServices is a programming interface providing access to many functions offered by Livelink Web Content Management Server™ (Livelink WCM Server for short). WCM WebServices makes it possible to access a Content server and thus the WCM-managed content from any computer. Access is completely platform-independent. For data exchange with the WCM system, the XML-based SOAP protocol is used.

WCM WebServices offers the following functions:

- developing separate WCM client applications tailored to your individual requirements
- creating dynamic content for your website, with the possibility to access the functionalities for managing website content. For implementing the dynamic content, you can use any programming language.
- integrating the functionalities for managing website content in desktop and Office tools whose scope of functionality can be extended with common programming languages.

WCM WebServices is based on the WCM Java API and works with the same or very similar methods for enabling access over the Internet.

Important! Incorrect use of the programming interface described in this manual may lead to errors in the WCM system including system crashes and data loss. Incorrect programming can also cause problems concerning performance and system resources. For this reason it is essential to test the developed software with regard to correctness, stability, robustness, and performance before putting it to productive operation.

Gauss Interprise AG cannot assume any liability for the correct functionality of the developed software. Our Professional Services Group can help you plan and implement solutions. This may help you avoid problems right from the start.

About this Document

This document addresses operators of a WCM system who want to configure access to the data of the WCM system via Web Services (service providers) and software developers who want to access the functionalities of the WCM system via WCM WebServices (service requesters).

For accessing WCM functionalities via WCM WebServices, appropriate knowledge of the way Livelink WCM Server works and its functions is required. A client application for using WCM functions can be implemented in different programming languages.

The contents of this manual are organized as follows:

- Chapter 2 “Concepts” explains the fundamental concepts of WCM WebServices and Livelink WCM Server.
- Chapter 3 “Using WCM WebServices” mainly addresses service providers. It describes how the WCM system has to be configured for using WCM WebServices. It also deals with the WCM WebServices description and the messages displayed in the case of errors.
- Chapter 4 “Access to Administration Data and Authentication” describes the methods for retrieving groups, roles, and users of the WCM system, for retrieving system information, and for authentication.
- Chapter 5 “Object Management” describes the WCM data types and methods for accessing WCM objects.
- Chapter 6 “Application Examples” contains some programming examples for WCM WebServices.

The programming interface described here is a component of Livelink WCM Server. In addition to this Programmer’s Manual, the following sources provide information:

- WCM Java API Programmer's Manual: This manual contains information on interfaces, classes, and methods of the Java programming interface (WCM Java API), which enables you to use the functionality of the WCM servers via external programs.
- Content Client User Manual: This manual provides detailed instructions on all tasks involved in editorial work on WCM-managed websites.

- Livelink WCM Server Administrator Manual: This manual describes the configuration and administration of WCM systems. Moreover, it contains a detailed description of the technical concepts of Livelink WCM Server.
- Livelink WCM Server Installation Manual: This manual describes the installation of the WCM system and provides information on configuring HTTP server and JSP engine.

Livelink WCM Server offers flexible programming interfaces for using the core functionality of Livelink WCM Server via external programs. The following table provides an overview of the different interfaces and the available functions.

Table 1 – Available functions of the Livelink WCM Server APIs

Functionality of the	WCM Java API	Remote API	Portal Manager API	WCM WebServices
`AdminHandler`	✓	✓	✓ read-only by means of `VipUserBean`	
`AttributeHandler`	✓	✓	✓ by means of `VipWebsiteBean`	✓ by means of `VipWebService_Port`
`EventDispatcher`	✓	✓		

Functionality of the	WCM Java API	Remote API	Portal Manager API	WCM WebServices
MailHandler	✓		✓ by means of VipEMailBean	
ObjectHandler	✓	✓	✓ by means of VipObject HandlerBean	✓ by means of VipWebService_ Port
ObjectLoader	✓	✓		
SystemHandler	✓	✓		
WorkflowHandler	✓		✓ by means of VipWorkflow Bean	✓ by means of VipWebService_ Port
DeploymentHandler	✓	✓		
ContextHandler	✓	✓		
LivelinkAdmin Handler	✓		✓ by means of LivelinkUser Bean	
LivelinkObject Handler	✓		✓ by means of LivelinkObject Bean	

Typographic Conventions

The following conventions are used in the text to draw attention to program elements, etc.:

Item	Font or Symbol	Examples
Program interface, such as menu commands, windows, dialog boxes, field and button names	*Menu → Entry*	*File → Create*
Paths to directories, file and directory names	**Drive:\Directory\File name**	**D:\WCM\ admin.bat**

Quotations from program code or configuration files	`Code quotations`	`<head>`
		`<title>heading </title>`
		`</head>`

Variables, i.e. placeholders for specific elements	{variable}	{WCM installation directory}

Important **information** and **warnings** are enclosed in gray boxes. Make sure to read such information to avoid losing data or making errors when using and managing WCM systems.

CHAPTER 2

Concepts

This chapter gives you a general overview of Web Services. It also introduces the WCM WebServices programming interface and the concepts of Livelink WCM Server.

Web Services

Web Services are an XML-based standard for interfaces. They allow the direct communication of different applications via the Internet.

A Web Service can be called by a person or by another Web Service that wants to use the service or function offered. Calling this service is independent of the underlying software infrastructure. The interfaces are defined and designed in a uniform standard. Thus, Web Services can be integrated on the basis of platform-independent technologies. For business applications, this implies that there can be a universal solution when it comes to integration software, the so-called middleware between two proprietary systems.

Web Service Architecture

The Web Service architecture is based on the interaction of the following three roles:

- the provider of a Web Service or service
- a service directory
- the service requester

These roles are linked with each other by certain actions, such as publishing, creating a request, or establishing a connection.

In a typical Web Service scenario, the provider of the Web Service wants to make a certain service available via the Internet. For this purpose, the service provider defines a formal description of the Web Service. The service requester requires this description for accessing the Web Service. The service provider can publish this description in a universal service directory. Thus, the Web Service can be found easily.

A service requester who wants to access a Web Service usually looks for the service or its description in the service directory. The service description may also be provided directly by the service provider. By means of this description, the service requester establishes a connection to the service provider, i.e. the provider's Web Service, thus accessing the desired service.

The following figure illustrates the interaction between the roles mentioned above and the individual actions.

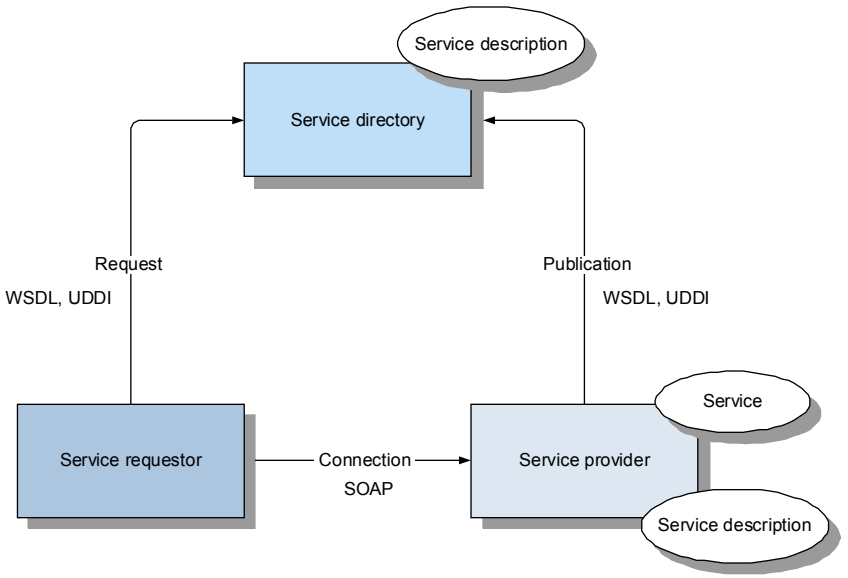


Fig. 1 – Roles and actions in the Web Service architecture

Standards for Web Services

The following standards basically determine the Web Services standard:

- SOAP
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery, and Integration)

In the following sections, these standards will be briefly described.

SOAP

SOAP is a standard that makes it possible to access Web Services, independent of the platform. It also defines the data transmission between service provider and service requester. The exchange format used is XML.

WSDL

WSDL (Web Services Description Language), the description language for Web Services, is an XML-based standard.

For an application to use Web Services, an exact “language regulation”, i.e. description, is required. The description must provide detailed information about the protocol used, the address and the port number, the available procedures and functions, as well as input and output formats. The provider of a Web Service makes this information available in the form of a WSDL file.

UDDI

The XML-based standard UDDI (Universal Description, Discovery, and Integration) specifies how detailed information on Web Services and their providers can be saved in a uniform way in directories.

UDDI directories are used to locate the many different Web Services. In these directories, providers can register their services and service requesters can make a targeted search for Web Services. Thus, a UDDI directory is a kind of (global) yellow pages. The Web Services are ordered according to certain properties. The respective entries reference the WSDL files of the associated Web Services.

In a UDDI directory, Web Services usually depend on business applications. Thus, the directory contains a categorization of industries and companies.

The Web Service Layer Model

The following figure illustrates the layers of the individual protocols.

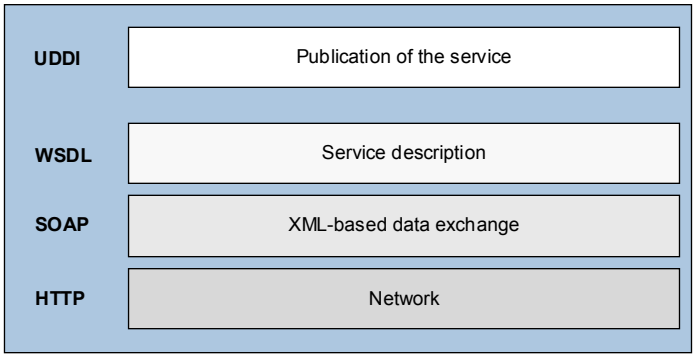


Fig. 2 – Protocol stack for Web Services

In the bottom layer of the protocol stack, the standard protocol HTTP is used for data transmission. Above this layer, XML data is exchanged on the basis of the SOAP standard. The standard describes the Web Service. In the topmost layer, the UDDI standard is used for publishing the service in a UDDI directory. This layer is optional. UDDI is not necessarily required for the functionality of Web Services.

Web Services Workflow

The following steps summarize how a Web Service is provided and used and which standards are deployed for this purpose.

1. The service provider creates a Web Service.
2. Using the standard WSDL, the provider supplies a description.
3. The service provider registers the Web Service with a UDDI directory. This procedure is optional.

4. Another Web Service or user finds the registered Web Service by searching a UDDI directory. This directory also provides the necessary description. Alternately, the service requester can get the description directly from the service provider.

   Afterwards, the service requester sends a request to the Web Service found.

5. The Web Service/user performing the request uses an application for establishing a connection to the respective Web Service. SOAP is the standard protocol used for accessing the Web Service.

6. For the actual data exchange, the exchange format XML and the protocol HTTP are used.

Using Web Services

The WSDL file contains the description of a Web Service. For interpreting this description, various toolkits are available to the service requesters. These toolkits read the description and automatically transform it into a module that can be used by a client application. The service requester – or client – integrates this module in the development environment and writes an own application in a programming language supported by the tool. The module works like a Web Service proxy that accepts objects and executes them on behalf of another application. This means that the Web Service client can address the actual Web Service by means of the module or the proxy and this way use the functions and methods offered by the Web Service. The respective objects are returned to the application via the module.

> **Note:** Service requesters who are familiar with SOAP do not necessarily need to use additional tools in order to create the HTTP requests required for accessing the Web Service.

The following figure illustrates the interaction of the individual components:

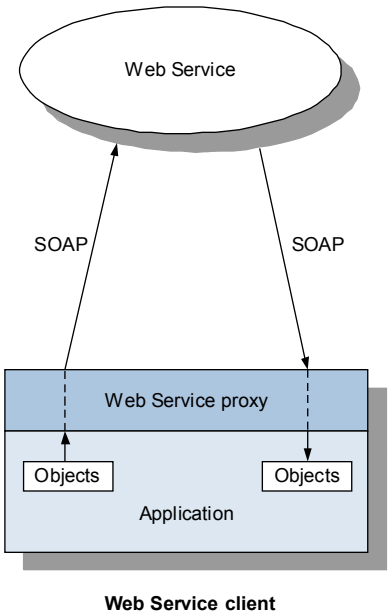


Fig. 3 – Method call of a Web Service client

WCM WebServices

WCM WebServices is a programming interface providing access to the functions and methods of Livelink WCM Server. WCM WebServices makes it possible to access a Content server and thus the WCM-managed content from any computer via the Internet. The methods made available this way can be called in any programming language.

WCM WebServices offers the following features:

- support of all data types used in Livelink WCM Server
- read and write access to the content of the WCM objects

- support of the staging concept of Livelink WCM Server
- complex search requests
- provision of Content Workflow to extend the three-stage staging concept by custom workflow steps

For **each** website, the Content server running in the context of a JSP engine or as web application in an application server provides a WSDL file. This file contains all data types and methods supported. For each website, there may be individually defined attribute sets. For this reason, there must be a WSDL file for each website.

The SOAP protocol used for exchanging messages consists of the following components:

- **SOAP envelope:** a kind of “envelope” describing what the message contains and what is to be done with it
- **SOAP body:** contains the actual utility data
- further information on transportation and code

The following figure illustrates the basic principle of WCM WebServices.

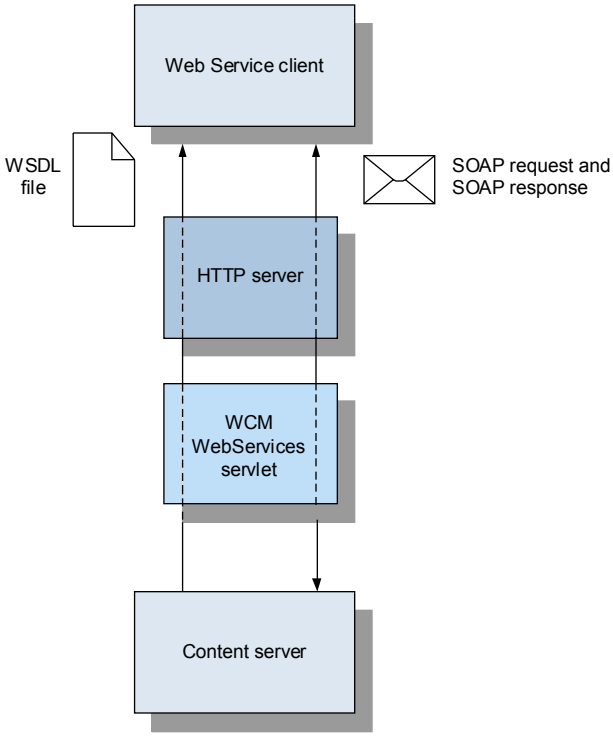


Fig. 4 – Basic principle of WCM WebServices

The Web Service client sends a SOAP request in a SOAP envelope to the Content server. The WCM WebServices servlet disassembles the SOAP envelope and passes the content of the message (SOAP body) to the WCM WebServices handler. After the message has been processed by the Content server, the servlet generates a message (SOAP response) with a similar structure and sends it back to the client. The module of the client receives the response, separates the SOAP elements and makes the utility data available as return values of the called function.

Note: For accessing the Content server via the WCM WebServices servlet, the respective user must be authenticated (basic authentication). This user must exist in the WCM system.

Concepts of Livelink WCM Server

The following section provides a short introduction to the basic concepts of Livelink WCM Server. The focus will be on the description of the terms that are relevant for working with WCM WebServices.

For a detailed description of the technical concepts and architecture of Livelink WCM Server, refer to the Livelink WCM Server Administrator Manual. For further information on working with the WCM system, refer to the Content Client User Manual.

Object Data

Livelink WCM Server is used for managing websites. Websites contain certain documents that are to be managed for different target groups. These documents may, for example, be HTML or XML documents, images, or Microsoft Word documents.

Documents managed by Livelink WCM Server are called WCM objects. Each WCM object has a certain type (see section “Object Type” on page 30) and consists of the following components, the so-called object data:

- **Content:** The content of a WCM object is the data from the actual document, such as the character sequence in an HTML file or the sequence of bytes of a graphics file.
- **Metadata:** An object’s metadata provide information on the document, e.g. its author, creation date, title, type, etc.

- **Log:** The log provides a description of all changes made to the object.

All WCM objects are referenced by their OID (Object Identifier). The OID identifies the *current* object.

In addition to a defined set of metadata, you can assign an attribute set to each object. An attribute set is a set of attributes that describe specific properties of object types. These may include the resolution of images or a copyright notice.

Moreover, you can categorize the content of WCM objects. For this purpose, you define object categories for which you can determine a number of properties. A possible object category would be “Invoice”, which could be characterized by the properties “Invoice_recipient” and “Status”.

Object Status

All WCM objects of a website managed with Livelink WCM Server pass through fixed stages: editing, quality assurance, and publication for productive operation. This so-called *staging* can be extended by assigning workflows to WCM objects. This enables you to realize successive editing by several editors or a multi-stage quality assurance.

Thus, the staging of Livelink WCM Server determines the status of a WCM object. According to the actions of object processing, the following object statuses exist:

- changed (also for new objects)
- checked out
- submitted
- rejected
- released

- delayed release
- deleted

As the result of the various work steps and actions performed on the WCM object, the status of an object changes. Each status change is logged. The WCM object's previous version is always archived in a separate data storage. This prevents older versions from getting lost. You can access these versions at any time.

Which actions can be performed on a WCM object depend on the following factors:

- the current status of the WCM object

  Note: The actions that can be performed on an object also depend on the status of the parent topic in the navigation topology.

- the access control list of the WCM object
- the functional areas assigned to the user

Object Type

The object type determines which document class a WCM object belongs to. This results in various WCM object properties, e.g.:

- attributes from an attribute set, which the WCM object might posses
- the type of templates the WCM object can use
- files that can be generated as a representation of the WCM object during page generation

In addition to object types, such as “HTML page”, “JSP template”, or “PDF document”, there are topic objects. A topic object can include other subordinate objects. It is thus used to structure the WCM objects in the navigation view and hence to structure the website. The template is another important object type. It determines the layout and is essential for generating pages.

Page Generation

A WCM object’s content differs from the data that is generated for an HTTP server. Each WCM object can have a template, which usually defines the layout for the web page to be generated. A template is a WCM object which can also contain a template. Among other things, the generated page depends on this template hierarchy (cascade).

Furthermore, the object type of the WCM object to be generated and the object types of the templates play an essential role in page generation. For GIF objects, for example, two files are created: the graphic as such and an HTML page with a reference to the graphics file. For HTML objects that have a template, the body element is cut from the HTML document.

Another decisive factor is the data storage view (Edit, QA, or Production). It determines which WCM objects are used for generating pages, see “Data Storage Views” on page 32.

Page generation is performed by the *deployment systems*. From the WCM objects stored in the database, the deployment systems generate the pages. By means of an HTTP server, these pages can be displayed in a browser. A deployment system generates the pages for a data storage view and a website. In a WCM system, you can configure any number of deployment systems.

Each page is referenced by a URL. As different pages can be generated by several deployment systems from the same WCM object, there are usually several URLs for the same WCM object.

Topologies – Organization of WCM Objects

Due to certain properties (metadata), a website's WCM objects can be displayed as a hierarchical structure with parent and child topics. In a website, WCM objects can be hierarchized according to two criteria: the topic structure (navigation topology) and the template structure (template topology).

Topic structure (navigation topology)

In the topic structure, objects are assigned topics. Each topic is itself a WCM object, which is the child object of another topic. This hierarchy is similar to the directories and subdirectories in a file system.

For each website there is exactly one topic that is the root object of the navigation tree of the website. Starting from the root object, the tree branches into topics and subtopics. All objects are directly or indirectly subordinate to this root object.

Template structure (template topology)

In the template structure, objects are organized according to their templates. Templates are also WCM objects. Each website can have one or more templates, which can be nested. All objects are subordinate to their respective templates. Objects that do not have any templates and are not of the object type "Template" are not represented in this view.

Data Storage Views

Creating and editing objects with Livelink WCM Server is subject to clearly defined stages composed of editing, quality assurance and publication for productive operation. According to the processing status of a WCM object, there are different views of the object: Edit, QA, and Production view.

- **Edit view:** In the Edit view, WCM objects, such as HTML pages, are added and edited by editors and graphics specialists. These users

need corresponding access rights for the objects. After they have been edited, the objects are submitted to Quality Assurance. This makes new objects or changes to existing objects also visible in the QA view.

- **QA view:** The QA view shows the WCM objects with all changes that have been submitted to Quality Assurance. Quality Assurance staff can check the changes with regard to content and formal aspects. On the basis of this check, they decide whether an object is to be sent back for correction or to be released. For release, the respective access right is required. After the release, the objects become visible in the Production view. Thus, the current version becomes available in the published website.
- **Production view:** The Production view makes the released pages of a website available to the user. By means of an HTTP server, these pages can be accessed via the Internet or intranet.

Depending on the object status, the WCM objects of a website are available in different views and can be edited. A new object, for example, is only available in the Edit view. It cannot be seen in the QA or Production view. Due to various actions that take place when objects are processed, the status of a WCM object can be different in the different views.

Authentication and Access Rights

For working with the WCM system, a user must provide authentication. When logging in, the user must enter a unique *user ID* and a password. The entries made are checked by the master Administration server of the WCM system.

Within a WCM system, there are different permissions. For working with WCM objects, object access rights and functional areas are of particular importance.

Access Rights for WCM Objects

In Livelink WCM Server, an *access control list* (*ACL*) is kept for each WCM object. This list defines the access rights for the respective users, groups, or roles (the so-called *principals*). Access rights can be explicitly marked as allowed or forbidden. Another possibility is not to set them explicitly. In the case of the latter, when checking rights, all assigned groups and roles are checked for an explicit permission or denial (in which case the denial always has priority).

The access control lists are inherited, i.e. each WCM child object in the navigation view normally inherits the rights of its parent object. When creating a website, a principal is determined, who initially has full access to all objects in the website (“website administrator”). The principal is added to the access control list of the website’s root object. All other WCM objects of this website inherit the rights of the root object, unless explicitly set otherwise.

Functional Areas

Each principal can be assigned functional areas. Via the functional areas, you control which functions of Livelink WCM Server are available to the user. The functional areas perform two important tasks:

- They determine which types of objects the users may add, check out, and check in. Some functional areas, such as “Basic”, are by default assigned to object types. Only users that have the corresponding functional area can add, check in, and check out objects of this object type.
- They determine which views and dialog boxes are available to the user in the Content client. This way, you can specify exactly which actions the user is allowed to perform.

Administration Rights for the WCM System

In addition to the access rights for WCM objects, there are administration rights for the WCM system. Principals with administration rights have access to the Admin client and can edit the elements of the WCM system, such as servers, websites, deployment systems, and user information. It is possible to assign graded access rights. Thus, a user may only be allowed to view and edit parts of the administration. The administration rights can only be changed via the Admin client.

CHAPTER 3

Using WCM WebServices

This chapter describes the necessary configuration for using WCM WebServices. It also deals with the WCM WebServices description and the messages displayed in the case of errors.

Configuration

To enable users to access your WCM system via WCM WebServices, the following configuration steps have to be performed:

- configuring a servlet mapping
- modifying the configuration of Livelink WCM Server

Notes:

If you create a web application (WAR file) via the Admin client for integrating the Content server in an application server, the necessary configuration is made automatically.

If you use MS Internet Information Server, enable the cache for the ISAPI applications for the website (for performance reasons).

Configuring the Servlet Mapping

In the configuration of the application server or JSP engine, configure a servlet mapping for the WCM WebServices servlet (class `de.gauss.vip.webservice.transport.WebServiceServlet`).

The mapping for application servers that comply with the servlet specification 2.2 requires the following entries in the file **web.xml**. This file is located in the **\WEB-INF** directory of the web application in which the Content server is started.

```
<servlet>
      <servlet-name>WebServiceServlet</servlet-name>
      <display-name>WCM WebServices-Servlet</display-name>
      <servlet-class>
         de.gauss.vip.webservice.transport.WebServiceServlet
      </servlet-class>
</servlet>
...
<servlet-mapping>
      <servlet-name>WebServiceServlet</servlet-name>
      <url-pattern>/WebService/*</url-pattern>
</servlet-mapping>
```

All requests starting with `/WebService/` will be forwarded to the WCM WebServices servlet.

Additional Steps for Resin

If you use Resin as your JSP engine, the following steps are additionally required:

1. In the file **{Resin installation directory}\conf\resin.conf**, enter the following system properties:

```
<system-property javax.xml.parsers.DocumentBuilderFactory=
   "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl"/>
<system-property javax.xml.parsers.SAXParserFactory=
   "org.apache.xerces.jaxp.SAXParserFactoryImpl"/>

<!-- xslt -->

<system-property javax.xml.transform.TransformerFactory=
   "org.apache.xalan.processor.TransformerFactoryImpl"/>
```

2. Copy the following JAR files from the directory **{WCM installation directory}\lib** to the directory **{Resin installation directory}\lib**:
   - **xalan.jar**
   - **xercesImpl.jar**
   - **xml-apis.jar**
   - **xmlParserAPIs.jar**

Modifying the Configuration

To enable users to access the Content server via WCM WebServices, the following changes must be made in the configuration:

- assigning the application `WebServiceApplication` to the Content server
- configuring the application `WebServiceApplication` by means of parameters

Use the Admin client to modify the configuration.

Notes:

The Content server must run in the context of a JSP engine or as a web application in an application server.

For detailed information on how to assign applications to a Content server and specify the respective parameters, refer to the Portal Manager API Programmer's Manual.

Assigning the Application `WebServiceApplication`

For the use of WCM WebServices, assign the application `WebServiceApplication` to the respective Content server. Start the Admin client of Livelink WCM Server. Make the assignment in the *Configuration* view via *Servers* → *{name of the Content server}* → *Applications*.

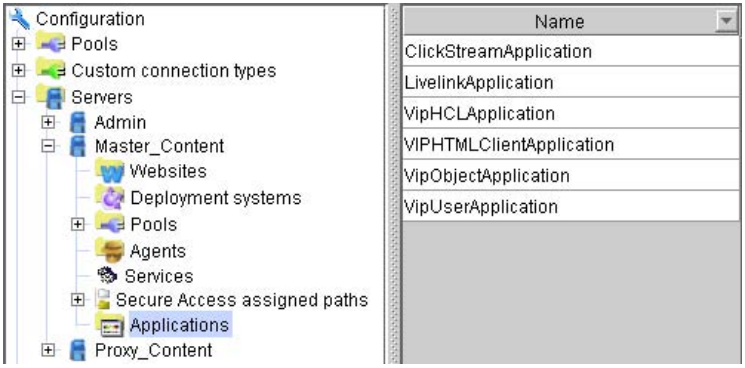


Fig. 5 – Applications of the Content server

Specifying the Parameters of the Application `WebServiceApplication`

To enable users to access the content of your WCM system via WCM WebServices, you have to explicitly allow access to the respective websites. This can be done by setting the parameters of the application `WebServiceApplication` accordingly.

The parameters are defined via *Configuration* → *Applications* → *WebServiceApplication* → *Parameters* → *{parameter name}*.

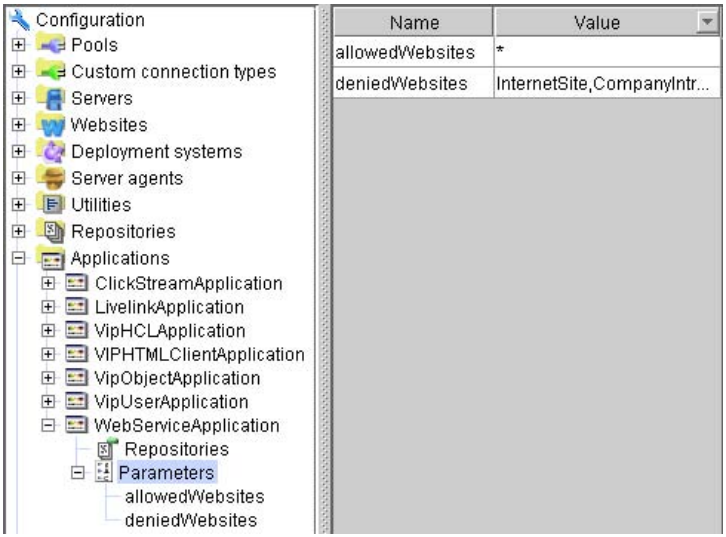


Fig. 6 – Parameters of the application `WebServiceApplication`

By default, the application `WebServiceApplication` has the parameters `allowedWebsites` and `deniedWebsites`.

- `allowedWebsites`: Enter a comma-separated list of website names. These websites can be accessed via WCM WebServices. Enter an asterisk (*) to allow access to all WCM-managed websites. The default value is empty, i.e. access via WCM WebServices is not possible.
- `deniedWebsites`: Enter a comma-separated list of website names. These websites **cannot** be accessed via WCM WebServices. Enter an asterisk (*) to deny access to all WCM-managed websites. A prohibition takes priority over a permission. The default value is empty.

The WCM WebServices Description

The description of WCM WebServices is provided in the form of a WSDL file. This section describes which information this file contains and how you can access this information.

Accessing the WCM WebServices Description

The WSDL file containing the WCM WebServices description is created dynamically. To access the file, enter the following URL in your browser or the tool you use:

```
{base URL of the HTTP server}/{directory of the web application}/
      WebService/Port/{website name}/{data storage view}/
      {name of the deployment system}?WSDL
```

Example

http://wcmserver.company.example/wcm/WebService/Port/InternetSite/edit/InternetSite_edit?WSDL

Explanation of the individual parts of the URL

- `?WSDL`: This parameter in the URL causes the generation of the WSDL file that is transmitted to the user/service requester.
- `{data storage view}`: Access to a website via WCM WebServices always refers to a specific data storage view. Possible values are:
  - Edit view: `edit`
  - QA view: `qa`
  - Production view: `prod`

  The different views of the website data are generated by the respective deployment systems.

  For generating the WSDL file, the information on the data storage view and on the name of the deployment systems are not necessarily required. If you do not specify the view of the website data, the Edit view data will be transmitted.
- `{name of the deployment system}`: Enter the name of the deployment system whose data you want to access. The deployment system must be installed on a Content server running in the context of a JSP engine or as a web application in an application server.

  If you do not specify a deployment system, the name is specified automatically. The Content server will look for a deployment system whose base URL is similar to the URL of the Web Service offered. This ensures that the Web Service client will get the URLs for accessing the generated pages, even if the deployment system is not specified.

  > **Note:** To precisely control access to the pages, you should explicitly specify the exact name of the deployment system with the desired data storage view in the URL.

Content of the WCM WebServices Description

The WSDL file for the description of WCM WebServices contains the following information:

- list of the available methods with required input parameters and return values
- WCM-specific data types
- URL the Web Service proxy uses for accessing the Content server

Functions and methods are called exclusively via remote procedure calls. The input parameters of each call contain the data that the Content server requires for the respective method calls. Afterwards, the Web Service proxy returns all data to the client via the return value of the method called.

The WCM WebServices description returns all WCM-specific data types that may be contained in the input parameters or in the return value. The most important data types are described in more detail in section “WCM-Specific Data Types” starting on page 72.

The Web Service proxy uses the following URL for accessing the Content server:

```
{base URL of the HTTP server}/{directory of the web application}/
      WebService/Port/{website name}/{data storage view}/
      {name of the deployment system}
```

Error Handling

The following elements are important for evaluating error situations:

- SOAP error messages
- error messages (*exceptions*) of the WCM servers

Structure of SOAP Error Messages

If the WCM server cannot successfully perform a method call, a SOAP error message is returned to the service requester. This SOAP error message contains a `Fault` element consisting of the following elements:

- `faultcode`: After the general identifier `Server`, this section contains the type of the error situation occurred, e.g. `AccessDeniedException`. The possible exceptions of the Content server are listed in table “Exceptions and possible causes” on page 47.
- `faultstring`: Description of the error situation and possibly the causes in plain text. This message is localized, i.e. it is output in the language selected for the logged-in user in the Admin client.
- `faultactor`: This element contains the identifier “Vip-Content-Manager-API”. Thus, the service requester knows that the exception occurred during processing in the WCM server and not during connection setup, transformation of the XML description of the input data, or transformation of the return values into a SOAP message.
- `detail`: This element contains a sequence of localized messages, which facilitate locating errors in the case of complex error situations.

The following example is a SOAP error message caused by the logged-in user “admin” not having the right to release a WCM object via `release`.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:tns="http://gaussvip.com/" xmlns:types="http://gaussvip.com/
     encodedTypes" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<soap:Fault>
<faultcode>tns:Server.AccessDeniedException</faultcode>
<faultstring>0474: The user &apos;admin&apos; does not have the
   right &apos;Release&apos;.</faultstring>
<faultactor>Vip-Content-Manager-API</faultactor>
<detail>
<tns:message sequenceNo="0">0303: The master server was unable to
    perform the request.</tns:message>
<tns:message sequenceNo="1">0635: Release of object
    &apos;171&apos; failed.</tns:message>
<tns:message sequenceNo="2">0300: Error while preparing for action
    &apos;Release&apos; on object &apos;171&apos; on website
      &apos;Susi&apos;.</tns:message>
<tns:message sequenceNo="3">0474: The user &apos;admin&apos; does not
    have the right &apos;Release&apos;.</tns:message>
</detail>
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

Exceptions of the Content Server

The error situations on the Content server are classified by the following exceptions:

Table 2 – Exceptions and possible causes

Name of the exception	Possible cause
`AccessDenied Exception`	The method called requires an action for which the logged-in user does not have the permission.
`ActionNotPermitted Exception`	Execution of a method is denied as this would lead to breaking a rule. This may, for example, be the case if a WCM object whose parent topic is not yet available in the QA view is to be submitted. Another reason may be that executing the method would lead to name conflicts.
`AttributeException`	This exception occurs if an unknown attribute is supplied, if the user wants to change an attribute the user is not allowed to change, or if a data type differing from the website configuration is used for the attribute.
`DatabaseException`	Problems occurred when accessing the RDBMS used by Livelink WCM Server.
`FileException`	Problems occurred during read or write access to a file.
`InvalidContextId Exception`	WCM WebServices manages a context ID for each logged-in service requester. If this ID becomes invalid, an exception is thrown. The service requester must re-login with user ID and password.

Name of the exception	Possible cause
`InvalidObject Exception`	If the attributes of a WCM object are edited almost simultaneously by two users, it might happen that the user whose change is processed last, internally starts the transaction with an outdated data set. This causes an exception. In the case of WCM WebServices, this is not very likely to happen as the internal object data is requested from the Content server only shortly before the change.
`LicenseException`	The licenses for Livelink WCM Server have expired.
`LoginException`	The service requester has logged in to the WCM system with an invalid user ID or wrong password.
`MailException`	Problems occurred while sending e-mails in the context of the staging, e.g. because the mail server could not be reached.
`NetException`	Problems occurred during usage of network connections by the WCM server, during connection setup, or during data exchange. Connection problems between the WCM WebServices requester and the application of the service provider are not the reason for this exception.
`ObjectInUseException`	The WCM object to be changed is currently being edited by another user of the WCM system.
`ObjectNotFound Exception`	The logged-in user does not have read rights for an object, the OID/version number entered when accessing a WCM object is invalid, or the respective object could not be found in the website.

Name of the exception	Possible cause
`RunlevelException`	The WCM server addressed or the website are in a run level in which the desired operation is not possible. The run level is changed, for example, for maintenance work on the WCM system.
`UnexpectedException`	During the execution of a method, internal errors occurred that cannot be attributed to any of the other error types. See also `VipApiException`.
`VetoException`	By means of a veto, an agent running on the WCM server prevented the method called from being executed. Agents can, for example, prevent certain staging actions from being performed on selected WCM objects in a certain period of time.
`VipApiException`	If an error cannot be attributed to any of the other error types, this exception is thrown.
`WorkflowException`	If a workflow transition has been selected that cannot be executed, this exception is thrown.

CHAPTER 4

Access to Administration Data and Authentication

By means of WCM WebServices, you can access the data and functions of the Administration server of the WCM system:

- retrieving profiles of users, groups, and roles with the respective settings and assignments (see section “Users, Groups, and Roles” on page 52)
- retrieving functional areas of principals (see section “Functional Areas” on page 59)
- retrieving website information (see section “Websites” on page 61)
- retrieving deployment system information (see section “Deployment Systems” on page 63)
- reading the system status (run level) of WCM servers and websites (see section “Run Levels” on page 64)
- retrieving other information of the WCM system, such as version and languages (see section “General Requests” on page 66)

Note: For accessing the data of the Administration server, administration rights are required, e.g. the right “Access to user administration” for reading profiles of users, groups, and roles, or the right “Access to system administration” for reading system information. Please refer to the Livelink WCM Server Administrator Manual for additional information on administration rights.

For using the functions of Livelink WCM Server via WCM WebServices, a login to the WCM system is required. Please also read section “Authentication” on page 67.

Users, Groups, and Roles

You can use WCM WebServices to read the settings (profiles) of users, groups, and roles and to set the default object rights of principals.

Data Types for Principals

For supplying the profiles of principals, the data types `User`, `Group`, and `Role` are used. All of these data types are based on the data type `Principal`.

The following table shows the components of the data type `User`.

Table 3 – Data types for users

Attribute	Data type	Description
`name`	`String`	User ID
`hasProfile`	`boolean`	Indicates whether further profile information is available
`commonName`	`String`	The user’s complete name
`eMailAddress`	`String`	The user’s e-mail address
`locale`	`Locale`	The user’s language
`websites`	`String[]`	List of the website names to which the user is assigned

Attribute	Data type	Description
subsitute	User	Substitute who has the right to log in on behalf of this user
substituteOf	User[]	Set of users for whom this user can act as a substitute
groups	Group[]	List of groups to which this user belongs
roles	Role[]	List of roles to which this user belongs
initially Granted Permissions	Permission[]	Default object rights granted to the user. These rights are entered as default settings when the user is added to the access control list (ACL) of a WCM object.
initially Denied Permissions	Permission[]	Default object rights denied to the user. These rights are entered as default settings when the user is added to the access control list (ACL) of a WCM object.
trustedLogin	boolean	Indicates whether trusted login is allowed for this user. This setting makes it possible that a user does not have to provide authentication for other products of Livelink WCM Server after having logged in to the system (single sign-on).
vipAccess	boolean	Indicates whether the user is allowed to access the WCM system

The data types Group and Role are structured identically. Their components are listed in the following table.

Table 4 – Data types for groups and roles

Attribute	Data type	Description
`name`	`String`	Unique name of the group or role
`hasProfile`	`boolean`	Indicates whether further profile information is available
`eMailAddress`	`String`	E-mail address of the group or role
`websites`	`String[]`	List of the website names to which the group or role is assigned
`members`	`User[]`	List of the users belonging to this group or role
`initially Granted Permissions`	`Permission[]`	Default object rights granted to the group or role. These rights are entered as granted when the group or role is added to the access control list (ACL) of a WCM object.
`initially Denied Permissions`	`Permission[]`	Default object rights denied to the group or role. These rights are entered as denied when the group or role is added to the access control list (ACL) of a WCM object.

Methods for Reading Profiles

The profile of a user, group, or role can be retrieved by means of the methods `getUserProfiles`, `getGroupProfiles`, and `getRoleProfiles`. These methods return an array of the data type `User`, `Group`, or `Role`. This array contains exactly those principals that meet the filter conditions specified.

For specifying the filter condition, data of type `Filter` and of the data types derived from this type are used. These data types are described in section “Searching for WCM Objects” starting on page 141.

The following restrictions apply to searches for users, groups, or roles:

- Only `EqualFilter`, `LikeFilter`, and `StringContainsFilter` can be used. For logical links, `AndFilter`, `OrFilter`, and `NotFilter` are allowed.
- Conditions can be specified for the attributes `name`, `commonName`, `eMailAddress`, `locale`, `trustedLogin`, and `vipAccess`.

`getGroupProfiles`

Returns the data of the groups matching the specified filter criterion. The result set can be restricted by `startResult` and `numberOfResults`.

Parameters

Table 5 – Parameters of the method `getGroupProfile`

Parameter	Data type	Description
`ldapContext`	`String`	LDAP context in which to perform the search, or `null` if the user administration is not based on LDAP
`sortList`	`Sort[]`	List of the attributes according to which the hitlist is to be sorted (in descending order of priority)
`filter`	`Filter`	The search criterion or `null` to search for all groups

Parameter	Data type	Description
`startResult`	`int`	Number indicating the first element of the search result 0 returns the search result from the beginning.
`numberOfResults`	`int`	Number of search results returned, starting with `startResult` `-1` returns all results.

Return

`Group[] groups`: list of groups matching the specified search criterion

`getRoleProfiles`

Returns the data of the roles matching the specified filter criterion. The result set can be restricted by `startResult` and `numberOfResults`.

Parameters

Table 6 – Parameters of the method `getRoleProfile`

Parameter	Data type	Description
`ldapContext`	`String`	LDAP context in which to perform the search, or `null` if the user administration is not based on LDAP
`filter`	`Filter`	The search criterion or `null` to search for all roles

Parameter	Data type	Description
sortList	Sort[]	List of the attributes according to which the hitlist is to be sorted (in descending order of priority)
startResult	int	Number indicating the first element of the search result 0 returns the search result from the beginning.
numberOfResults	int	Number of search results returned, starting with startResult -1 returns all results.

Return

Role[] roles: list of roles matching the search criterion specified

getUserProfiles

Returns the user data of the users matching the specified filter criterion. The result set can be restricted by startResult and numberOfResults.

Parameters

Table 7 – Parameters of the method getUserProfile

Parameter	Data type	Description
ldapContext	String	LDAP context in which to perform the search, or null if the user administration is not based on LDAP
filter	Filter	The search criterion or null to search for all users

Parameter	Data type	Description
sortList	Sort[]	List of the attributes according to which the hitlist is to be sorted (in descending order of priority)
startResult	int	Number indicating the first element of the search result 0 returns the search result from the beginning.
numberOfResults	int	Number of search results returned, starting with startResult -1 returns all results.

Return

User[] users: list of users matching the search criterion specified

Supplying Default Object Rights

The component initialGrantedPermissions of the data types User, Group, and Role contains a list of default object rights granted to the principal. These default access rights should be suggested as setting by a WCM WebServices utility when the principal is added to the access control list of a WCM object. The component InitialDeniedPermissions is used analogously and contains the default object rights denied to the principal.

The access control list of a WCM object is changed by means of the data type Acl, see “Changing the ACL of a WCM Object” on page 87.

Functional Areas

In Livelink WCM Server, users must be directly or indirectly (i.e. due to their membership in groups or roles) assigned to functional areas in order to perform certain actions in the context of website management. For general information on functional areas, see section “Functional Areas” on page 34.

In WCM WebServices, functional areas are represented by the data type `FunctionalArea`. The method `getFunctionalAreas` returns all functional areas defined in the WCM system.

The data type `FunctionalArea` contains the name of the functional areas and the sets of users, groups, and roles **directly** assigned to this functional area.

The following table shows the available default functional areas and the associated names. Via the Admin client, you can define custom functional areas and assign them to object types. These functional areas are represented by their names.

Table 8 – Names of the default functional areas

Functional area	Constant	Allowed view or function in the Content client
Basic	`VIP`	Create, check out, and check in objects based on these object types
Advanced	`ADVANCED`	
Dynamic	`DYNAMIC`	
Form	`FORM`	
Workflow	`WORKFLOW`	Assign workflows to objects, remove workflow assignments

Functional area	Constant	Allowed view or function in the Content client
Intelligent templates	ITF	Not used by default, required for compatibility with VIP 5e
Direct release	DIRECT_RELEASE	Edit the option *Direct release* in the metadata
References dialog	REFERENCES	View the References dialog box
Access rights dialog	ACCESS_RIGHTS	View the Access rights dialog box
Log dialog	LOG	View the Log dialog box
Livelink	LIVELINK	View the Livelink metadata dialog box
Filter standard	FILTER_STANDARD	Use the standard filters
Filter edit	FILTER_EDIT	Create and edit filters in the filter editor
View Subordinate objects	OBJECTLIST	“Subordinate objects” view
View Object list	LISTVIEW	“Object list” view
View My objects	FILTER_TODO	“My objects” and “My work list” views
View Template structure	TEMPLATE_STRUCTURE	“Template structure” view
Import	IMPORT	Use import functions
Search	COMI_SEARCH	Use search functions

Methods for Reading Functional Areas

`getFunctionalAreas`

Returns all functional areas available in the WCM system

Parameters

none

Return

`FunctionalArea[] functionalAreas`: the available functional areas

Websites

A website is identified by a unique name within a WCM system. WCM WebServices offers the method `getWebSiteNames` for reading the names of all websites managed in the WCM system. Usually, each website is assigned several deployment systems.

Methods for Websites

`getAttributeSets`

Returns all attribute sets assigned to the website

Parameters

none

Return

`AttributeSet[] attributeSets`: the available attribute sets

getObjectCategories

Returns all object categories assigned to the website

Parameters

none

Return

`ObjectCategory[] objectCategories`: the available object categories

getObjectTypes

Returns all object types assigned to the website

Parameters

none

Return

`ObjectType[] objectTypes`: the available object types

getWebsiteNames

Returns the names of all websites available in the WCM system

Parameters

none

Return

`string[] websiteNames`: the names of the available websites

Deployment Systems

There are three different kinds of deployment systems which correspond to the three data storage views: Edit, QA, and Production. A deployment system for the Edit view has access to all versions of the WCM objects and uses them to generate pages. In the Quality Assurance or Production deployment systems, only the relevant data of the respective view is available.

The essential properties of a deployment system are:

- the website on which the deployment system is based
- the server on which the deployment system is executed
- the directory on the above server where the generated pages are to be stored
- the base URL which is to be used to generate references

Use the `getDeploymentSystems` method to retrieve a list of all deployment systems for a website. For each deployment system, the name and type are returned to the service requester.

`getDeploymentSystems`

Returns all deployment systems for the specified website

Parameters

Table 9 – Parameter of the method `getDeploymentSystems`

Parameter	Data type	Description
`website`	`String`	Name of the website

Return

`DeploymentSystem[] deploymentSystems`: list of the available deployment systems

Run Levels

The run level describes the current state of a server or a website. This state defines which of the server's or website's functional components are currently activated or deactivated. The run levels are divided into server run levels and website run levels. The website run levels are based on the server run levels. For further information on run levels, refer to the Livelink WCM Server Administrator Manual.

The run levels can be retrieved by means of the method `listAllRunlevels`. When a server starts, the run levels progress from run level `0` to run level `10`, when the servers shuts down, from `10` to `0` accordingly. The run levels of servers and websites can also be controlled separately via the Admin client of Livelink WCM Server.

Use the `getRunlevel` method to retrieve the current run level of the website assigned to the Web Service (i.e. the run levels from 6 upwards).

The following run levels are available for servers:

- **0 Server not available:** The server is not available.
- **1 Connections closed:** All communication connections are closed. Database and LDAP connections, as well as services are no longer available. When changing to the next higher run level, connection management is established.
- **2 No users logged in:** No users are logged into the server. When changing to the next higher run level, user administration is activated.
- **3 No agents running:** All server agents are shut down. When changing to the next higher run level, all server agents start.

- **4 Single user mode:** All users except the administrator who changes the run level are logged out and can no longer log in to the WCM system. This run level is especially designed for maintenance work on the WCM system. When changing to run level 3, the websites will be shut down completely. When changing to the next higher run level, the system becomes available for all users.
- **5 Server up:** Startup of the server is complete.

The following website run levels exist. For the website run levels, server run level 4 is required at least.

- **6 Website inaccessible:** One or all websites are no longer accessible (not even for read-only access). This means that the respective deployment systems are also no longer active.
- **7 Configuring website:** The specified website is configured in this run level (the configuration data is written). When changing to the next higher run level, the websites are initialized for read-only access.
- **8 Website read only:** One or all websites are available for read-only access. Data depending on the deployment system, such as URLs and paths, are not yet available in this run level. When changing to the next higher run level, the deployment systems are initialized.
- **9 Deployment complete:** Initialization of the deployment systems is complete, i.e. deployment jobs are in progress. On transition to the run level 8, no more deployment jobs are processed. On transition to the next higher run level, the websites are released for write access. In the next higher run level, the WCM objects are distributed to proxy Content servers.
- **10 Website up:** One or all websites are available for write access.

Methods for Reading Run Levels

`getRunlevel`

Returns the run level of the website assigned to the Web Service

Parameters

none

Return

`Runlevel runlevel`: the current run level of the website

`listAllRunlevels`

Returns all run levels possible in a WCM system

Parameters

none

Return

`Runlevel[] runlevels`: the possible run levels

General Requests

For general requests, the following methods are available:

`getLanguages`

Returns all languages supported by the server

Parameters

none

Return

`Locale[] languages`: list of the available languages

`getVIPVersion`

Returns the version of Livelink WCM Server

Parameters

none

Return

`KernelVersion version`: version of Livelink WCM Server

Authentication

Accessing a WCM system via WCM WebServices requires a user authentication. Otherwise, the individual methods cannot be used. For many actions, the user's functional areas or the access control lists of the WCM objects are checked to determine whether the user has the necessary rights.

Authentication is performed when the Web Service registers with the WCM server. In the WCM system, a `ContextId` object is created, which differs for each login and is unique within the WCM system. The Web Service is thus uniquely registered with the system. The `ContextId` remains valid in the WCM system as long as the session belonging to the current Web Service exists.

For WCM WebServices, no additional `login` method is required. The user name and password for login to the WCM system are supplied via the application programmed on the basis of WCM WebServices. Name and password must be entered in the header of the HTTP request.

The following methods are available for terminating a session, authenticating users as substitutes, and changing passwords:

changePassword

Changes the password for the specified user

Parameters

Table 10 – Parameter of the method changePassword

Parameter	Data type	Description
newPassword	String	New password

Return

none

logout

Logs out a user known by the user's ContextId

Parameters

none

Return

none

substituteLogin

Logs in the already logged-in user – known by the user's ContextId – to the WCM system as the substitute of the user specified by means of userName. The user represented by the ContextId must be allowed to work as the substitute of the specified user.

Parameters

Table 11 – Parameter of the method `substituteLogin`

Parameter	Data type	Description
`userName`	`String`	ID of the user on whose behalf the logged-in user wants to work

Return

none

CHAPTER 5
Object Management

This chapter provides a description of the most important data types used for editing WCM objects via WCM WebServices. Afterwards, the object management methods are introduced. The following main groups are distinguished:

- general parameters such as `DeploymentWaitInfo` and `EMailInfo`, see section “General Parameters” on page 93
- methods for executing staging actions such as “check out” or “submit”, see section “Staging Methods” on page 95
- methods for executing Content Workflow, see section “Workflow Methods” on page 104
- methods for editing WCM objects, e.g. for changing the metadata, see section “Object Management Methods” on page 114

> **Note:** As the syntax of the method call depends on the programming language used, a general description of method names and parameters will be provided.

The last section describes how to search for WCM objects by means of filters, see section “Searching for WCM Objects” on page 141.

WCM-Specific Data Types

Each WCM object is represented by a number of data. To access the WCM system via WCM WebServices, you must know the data types that are used. A data type determines the possible values and their use.

In Livelink WCM Server, the data – with the exception of integer and Boolean values – is internally characterized by Java classes/Java interface definitions. In WCM WebServices, these data types are assigned XML schema data types. Partly, the standard XML schema “Simple Types” can be used. For other WCM data types, WCM-specific “Complex Types” must be used, which belong to the namespace “http://gaussvip.com/”.

The WCM WebServices WSDL file provides a complete overview of the data types used, see “Content of the WCM WebServices Description” on page 44. In the following, the most important data types are described:

- OID and versions of WCM objects, see “Data Type: `ObjectId`” on page 73
- metadata, see “Data Type: `ObjectData`” on page 74
- object type, see “Data Type: `ObjectType`” on page 80
- object status, see “Data Type: `ObjectState`” on page 81
- object access rights, see “Data Type: `Permission`” on page 84
- access control list, see “Data Type: `Acl`” on page 86
- entry in the access control list, see “Data Type: `AclEntry`” on page 87
- representation of tree structures for objects, see “Data Type: `MultiImportPart`” on page 88
- workflow actions, see “Data Types for Workflow Actions” on page 89

Data Type: ObjectId

The object ID, the so-called OID, represents the reference to a WCM object. It is a unique identifier for a WCM object within a website. The uniqueness only refers to the current version of a WCM object. Archived objects have the same OID with a different version number.

Table 12 – The components of the data type ObjectId

Attribute	Data type	Description
id	String	Internal representation as a character string

If a WCM object goes through a staging step, such as editing or Quality Assurance, a new version of the object is created. For versioning WCM objects, the data type Version is used. A version consists of three numbers: a major version, a minor version, and a micro version number. The micro version number changes when a WCM object is changed in the Edit view (check out, change metadata). The minor version number changes when a WCM object is submitted to Quality Assurance; in this case, the micro version number is set to 0. The major version number is incremented at release; in this case, the minor version number is set to 0. By means of the method getVersionList, you can determine the available versions of a WCM object, see “getVersionList” on page 135.

Data Type: `ObjectData`

For general information on object data, refer to section “Object Data” on page 28.

The data type `ObjectData` contains the metadata of a WCM object. When searching with the `filter` method, an `ObjectData` array is returned which contains one element for each WCM object found. These elements contain exactly the attributes specified in the input parameter `attributeKeys` of the `filter` method.

Use the `get` method to retrieve this data from the server for exactly one WCM object in a specific version. The desired attributes must be selected first in the `attributeKeys` list.

For the `change` and `create` methods, a parameter of the type `ObjectData` is passed to the server for changing the attributes or for setting the attributes of a new object.

The icons in the following table illustrate the possible actions for the respective metadata item:

The value `null` is allowed for this attribute.

The attribute can be searched for, i.e. it can be used for constructing filters.

The attribute value can be changed.

Table 13 – The components of the data type `ObjectData`

Attribute	Data type	Description
`ACL`	`Acl`	Access control list of the WCM object
`attributeKeys`	`String[]`	If the object data is sent to the server with a `create` or `get` method, this array must contain the set of attribute names whose values are to be set or changed by the server. When the object data is returned by the server, this array contains the set of attribute names that were actually set by the server. All other attributes have non-reliable values.
`children`	`ObjectId[]`	OIDs of the WCM object's subordinate objects (children) if the WCM object is a topic. If the WCM object is not a topic or if it does not have any subordinate objects, this value is `null`.
`compatible ObjectTypes`	`ObjectType[]`	Object types into which the object type of the WCM object can be changed
`contentSize`	`int`	Size of the object content in bytes `-1` means that the object does not have any content
`createdBy`	`User`	User who created the WCM object

Attribute	Data type	Description
createdDate	dateTime	Date (with time) on which the WCM object was created
deploymentHint	String	Deployment hint, i.e. the suggested name for the page generated from the WCM object
description	String	Description of the object content
directRelease	boolean	This value is true if the WCM object can be released directly without having to go through Quality Assurance.
editEMail Receivers	String[]	E-mail addresses that are to be used for notification if the WCM object is rejected or reaches its expiration date
expireDate	dateTime	Date (with time) until which the WCM object remains valid
keywords	String[]	Set of keywords assigned to this WCM object
linkedFrom	ObjectId[]	Array of OIDs of the WCM objects containing references to the current WCM object. If no references exist, `null` is returned.

Attribute	Data type	Description
`linksTo`	`Link[]`	Array of references contained in the WCM object's content. If no references exist, an empty array is returned.
`locale`	`Locale`	Language used for the WCM object (locale)
`modifiedBy`	`User`	ID of the last user who changed the WCM object
`modifiedDate`	`dateTime`	Date (with time) on which the WCM object was changed last
`objectCategory`	`String`	Name of the WCM object's object category or `null` if the WCM object is not assigned to an object category
`objectId`	`ObjectId`	OID that clearly identifies the WCM object in a website
`objectState`	`ObjectState`	Current status of the WCM object
`objectType`	`ObjectType`	Object type of the WCM object. The object type can be changed according to the rules for compatible object types.

Attribute	Data type	Description
pathname	String	Path name of the generated page of the WCM object. The path name depends on the deployment system. If there is no deployment system for this website, this value is null.
pendingReleaseDate	dateTime	Date (with time) at which the object version is to be published in the Production view
possibleActions	String[]	Possible staging actions that can be performed with the current status of the WCM object and the rights of the user
QAEMailReceivers	String[]	E-mail addresses that are to be used for notification if the WCM object is submitted to Quality Assurance
releasedBy	User	ID of the last user who released the WCM object
releasedDate	dateTime	Date (with time) on which the WCM object was released last
releaseEMailReceivers	String[]	E-mail addresses that are to be used for notification if the WCM object is released

Attribute	Data type	Description
`subtitle`	`String`	Heading of the WCM object
`surrogateURL`	`String`	URL for the surrogate page of the WCM object or `null` if this object type does not require a surrogate page
`targetGroup`	`String`	Target group of the WCM object
`template`	`ObjectId`	OID of the WCM object's template or `null` if the object does not have a template
`title`	`String`	Title (name) of the WCM object
`topic`	`ObjectId`	OID of the topic in which the WCM object is located or `null` if the WCM object is the root object of the website
`URL`	`String`	URL for the generated page of the WCM object. The URL depends on the deployment system to which WCM WebServices was assigned. If no deployment system exists, `null` is returned.

Attribute	Data type	Description
version	Version	Current three-digit version number of the WCM object
websiteName	String	Name of the website to which the WCM object belongs

Data Type: ObjectType

For general information on object types, refer to section “Object Type” on page 30.

The data type ObjectType represents the type of a WCM object. Object types are website-specific and are thus only valid within a specific website. Use the getObjectTypes method to retrieve the object types of a website (see “getObjectTypes” on page 62). Use the change method to change the object type (see “change” on page 117). The attribute compatibleObjectTypes of the data type ObjectData provides an overview of compatible object types.

Table 14 – The components of the data type ObjectType

Attribute	Data type	Description
attributeSetName	String	Name of the attribute set assigned to this object type
defaultSuffix	String	Default extension of files containing content of WCM objects of this type. This extension is, for example, used when new WCM objects are created.
description	String	Localized description of the object type

Attribute	Data type	Description
`fileOnCreate Needed`	`boolean`	This value is true if a file must be specified when a WCM object of this type is created.
`frame`	`boolean`	This value is true if the object type is a frame.
`imageURL`	`String`	URL to the file of the icon representing the object type. The URL depends on the deployment system assigned to WCM WebServices. If there is no deployment system for this website, `null` is returned.
`mimeType`	`String`	MIME type belonging to the object type
`name`	`String`	Name of the object type
`template`	`boolean`	This value is true if the object type is a template.
`topic`	`boolean`	This value is true if the object type is a topic.

Data Type: `ObjectState`

For general information on the status of WCM objects, refer to section “Object Status” on page 29.

The data type `ObjectState` represents the status in the life cycle of a WCM object.

Table 15 – The individual object statuses

Status	Description
checked_out	The WCM object has been checked out.
deleted	The WCM object has been deleted.
edited	The WCM object has been changed (metadata or content).
rejected	The WCM object has been rejected.
released	The WCM object has been released.
release_at	The WCM object has been released, but will be published later.
submitted	The WCM object has been submitted to Quality Assurance.

The following table shows the statuses and the possible transitions between them. Transition to another status takes place in the context of staging actions. When a new WCM object is created, it acquires the status “edited”. For information on the individual staging methods, refer to section “Staging Methods” starting on page 95.

Table 16 – The possible transitions between the object statuses

Status before action	Status after action	Action
edited	edited	change
	checked_out	checkOut
	deleted	delete
	submitted	submit

Status before action	Status after action	Action
checked_out	edited	checkIn
	Back to the status the object had before	undoCheckOut
deleted	Object is finally deleted from the WCM system.	destroy
	rejected	reject
submitted	rejected	reject
	released	release
	pending_release	release
pending_release	released	Automatically when the publication date is reached
	checked_out	checkOut
	edited	change
released	checked_out	checkOut
	edited	change

Table 17 – The components of the data type `ObjectState`

Attribute	Data type	Description
`description`	`String`	Localized description of the object status
`imageURL`	`String`	URL to the file of the icon representing the object status. The URL depends on the deployment system assigned to WCM WebServices. If there is no deployment system for this website, `null` is returned.
`name`	`String`	Name of the status

Data Type: `Permission`

For general information on access rights, refer to section “Access Rights for WCM Objects” on page 34.

The data type `Permission` represents an access right which can be denied or granted for WCM objects. Each WCM object has an access control list (ACL) containing a set of principals together with the access rights granted and denied to these principals. The WCM object inherits the ACL of the parent object (recursively) if it does not have an ACL of its own. For further information on ACLs, see the section on “Data Type: `Acl`” on page 86.

Table 18 – Possible access rights for WCM objects

Name	Abbreviation	Description
CHANGE_RIGHTS	cr	Right to change the access rights of a WCM object
CREATE	c	Right to create a new WCM object
DELETE	d	Right to delete WCM objects
READ	r	Right to read a WCM object
READ_PRODUCTION	rp	Right to read a WCM object in the Production view
RELEASE	rl	Right to release a WCM object
TREE_OPERATIONS	t	Right to copy and move a WCM object
WRITE	w	Right to change a WCM object
WRITE_META	wm	Right to change metadata

Table 19 – The components of the data type `Permission`

Attribute	Data type	Description
name	String	Name (as abbreviation) of the access right

Data Types for Access Control Lists

Two data types are available for editing access control lists:

- `Acl`
- `AclEntry`

Data Type: `Acl`

An access control list (= ACL) combines a set of access rights. In a WCM system, an ACL typically contains all access rights that have been set for a WCM object.

In an ACL, the access rights for principals (users, groups, roles, group-roles, or “World”) are set. An access right can be granted or explicitly denied.

Table 20 – The components of the data type `Acl`

Attribute	Data type	Description
`entries`	`AclEntry[]`	Array of access rights granted or denied to the given principal
`inherited`	`boolean`	This value is true if the ACL object is inherited, i.e. if the access rights are adopted from the parent object. The object to which the inherited ACL belongs is available via the attribute `owner`.
`owner`	`ObjectId`	OID of the WCM object to which this ACL belongs

Data Type: AclEntry

This data type represents the assignment of a principal to a granted or denied access right.

Table 21 – The components of the data type AclEntry

Attribute	Data type	Description
permission	Permission	Access right assigned to the principal by this AclEntry
policy	boolean	Defines whether the access right is to be granted (true) or denied (false) to the principal
principal	Principal	Associated principal

Changing the ACL of a WCM Object

How to change the ACL of an WCM object:

1. For each principal to be included in the new ACL, generate a respective User, Group, Role or GroupRole object.
2. For each type of right to be defined in the ACL, create a respective Permission object.
3. With the created Principal and Permission objects, generate an AclEntry object for each right to be granted or denied. In this AclEntry object, the attribute policy controls whether the right is granted (true) or denied (false).
4. Create an array of AclEntries to which the AclEntries just generated are written.
5. With this field, the attribute entries of an Acl object is set. The attributes owner and inherited are irrelevant in this case.

6. Set the following attributes of an `ObjectData` object to the following values:
   - set the `ACL` attribute to the generated `Acl` object
   - set the `ObjectId` to the OID of the WCM object to be changed
   - set the array `attributeKeys` to the attribute name `ACL`
7. With this `ObjectData` object, call the method `change` to set the new rights for the WCM object, see “change” on page 117.

Data Type: `MultiImportPart`

This data type is used to represent a tree-like structure of WCM objects that can be created by Livelink WCM Server with the method `multiImport` (see “multiImport” on page 137).

Table 22 – The components of the data type `MultiImportPart`

Attribute	Data type	Description
`children`	`MultiImportPart[]`	Defines the WCM objects to be created below the current WCM object. This list is only evaluated if the object to be created is of type “Topic”.
`content`	`base64Binary`	Content of the WCM object to be created. `null` is allowed if the object is to be created without content. Regarding the content, the rules for the respective object type apply: an object of type “Frame” must, for example, contain a `frameset` tag.

Attribute	Data type	Description
`objectData`	`ObjectData`	Defines the metadata of the object to be created. The following fields must be set: `deploymentHint`: The suggested file name of the generated page `objectType`: The object type of the WCM object to be created `title`: The title of the WCM object `attributeKeys`: The list of names of the attributes that are set (at least `deploymentHint`, `objectType`, `title`)

Data Types for Workflow Actions

In Livelink WCM Server, the predefined stages (editing, quality assurance, publication for productive operation) can be extended by user-defined workflow steps in order to realize editing by several editors or a multi-stage quality assurance process.

A workflow definition may contain any number of activities and transitions. Each activity is assigned to a principal (user, group, or role) who may edit the object and forward it in the workflow. The three types of activities (Edit, QA, and Delete) represent the task of the assigned principal. By means of the transitions, the WCM object is forwarded from one activity to the next. The necessary staging transitions are automatically performed in the background.

Four data types are available for working with Content Workflow:

- `Workflow`
- `Activity`

- Transition
- WorkflowNavigationInfo

Data Type: Workflow

An object of type “Workflow” represents a so-called workflow definition, which contains the individual workflow steps. Workflow definitions are created and edited with the Content Workflow Modeler. For creating objects of the type “Workflow”, the user requires the functional area “Workflow”.

Table 23 – The components of the data type Workflow

Attribute	Data type	Description
activities	Activity[]	Individual activities of a workflow
assignedObject	ObjectId	WCM object assigned to the workflow
currentActivities	Activity[]	All activities of a workflow
currentActivity	Activity	Current activity
description	String	Description of the workflow object
hasBeenForwarded	Boolean	Retrieves whether the assigned workflow objects has already been forwarded
hasContent	Boolean	Retrieves whether die XPDL file on which the workflow definition is based has any content
name	String	Name of the workflow object
processId	String	Process ID
startActivities	Activity[]	All start activities of the current workflow object

Attribute	Data type	Description
workflowOid	ObjectId	OID of the workflow object
workflowTitle	String	Title of the workflow object
workflowVersion	Version	Version of the workflow object

Data Type: Activity

The individual activities of a workflow are represented by the Activity interface. The activity type represents the staging of Livelink WCM Server. For further information on the use of activities, refer to the WCM Java API Programmer's Manual.

Table 24 – The components of the data type Activity

Attribute	Data type	Description
activityType	String	Staging type of the activity
description	String	Description of the activity
id	String	ID of the activity
name	String	Name of the activity
principalName	String	Name of the principal assigned to the activity
principalType	int	Type of the principal assigned to the activity
transitions	Transition[]	All transitions of the activity

Data Type: Transition

A targeted transition between two workflow activities is represented by the Transition interface. In addition to the name and the description of the transition, the activities assigned to a transition can be retrieved.

Table 25 – The components of the data type Transition

Attribute	Data type	Description
description	String	Description of the transition
fromActivity	Activity	Activity to be used as the source of the transition
id	String	ID of the activity
toActivity	Activity	Activity to be used as the target of the transition
name	String	Name of the transition

Data Type: WorkflowNavigationInfo

The WorkflowNavigationInfo object functions as a container collecting information on workflows. Such information can be used for developing user interfaces for workflows.

Table 26 – The components of the data type WorkflowNavigationInfo

Attribute	Data type	Description
activity	Activity	Possible activities of a principle
workflow	Workflow	Workflow definition

General Parameters

Some of the methods described in the following use the parameters `DeploymentWaitInfo` and `EmailWaitInfo`. These parameters always have the same function and are thus only described once here.

The `DeploymentWaitInfo` Parameter

The data type `DeploymentWaitInfo` represents the information used to wait for all deployment jobs belonging to an action. This data type is used in procedure calls that trigger staging actions.

If the parameter `DeploymentWaitInfo` is not `null`, the calling method waits for the deployment systems specified in this parameter to finish all associated deployment jobs. The maximum waiting time is defined by `DeploymentWaitInfo.timeout`. The method returns, at the latest, if the waiting time for this action exceeds the timeout value specified.

If, in a staging method call, the parameter `DeploymentWaitInfo` is `null`, the method does not wait for any deployment system. In this case, the deployment jobs will be processed asynchronously.

Table 27 – The components of the data type `DeploymentWaitInfo`

Attribute	Data type	Description
`deploymentSystems`	`String[]`	List of deployment systems to be waited for
`isTimeout`	`boolean`	In WCM WebServices, this field is reserved for future functions. At present, it is not considered.
`timeout`	`long`	Maximum time (in milliseconds) that the method waits for the deployment jobs to finish

The `EMailInfo` Parameter

The data type `EmailInfo` represents the information used for automatically creating e-mails. These e-mails are sent when WCM objects are submitted, released, deleted, or rejected. The e-mails can be sent by the WCM server if the respective methods are used.

The algorithm implemented in the WCM server collects all e-mail addresses from the data of the WCM object. The action defines which attributes are used for creating the e-mail addresses. The relevant attributes are `editEMailReceivers`, `QAEMailReceivers`, and `releaseEMailReceivers`.

If, in the call of a workflow method, the parameter for the `EmailInfo` is null, the WCM system does not generate any e-mails.

Table 28 – The components of the data type `EMailInfo`

Attribute	Data type	Description
`cc`	`String`	CC field of the e-mail for entering additional recipients
`excludeOIDList`	`boolean`	This value is true if the list of the OIDs is not to be included in the generated e-mail.
`subject`	`String`	Subject of the e-mail
`text`	`String`	Content of the e-mail

Staging Methods

This section introduces the methods of WCM WebServices that can be used for performing staging actions.

- checking out objects for editing, see “checkOut” on page 96
- undoing check out of objects, see “undoCheckOut” on page 103
- checking in objects after editing, see “checkIn” on page 95
- releasing objects directly without submitting them, see “directRelease” on page 98
- submitting objects to Quality Assurance, see “submit” on page 101 and “submitImmediately” on page 102
- rejecting objects in Quality Assurance, see “reject” on page 99
- releasing objects in Quality Assurance, see “release” on page 100

In the following overview, the methods are sorted alphabetically.

checkIn

Returns the (modified) content of the specified WCM object to the WCM system

Parameters

Table 29 – Parameters of the method checkIn

Parameter	Data type	Description
oid	ObjectId	OID of the respective WCM object
content	byte[]	Content of the WCM object, specified as a file

Parameter	Data type	Description
keepCheckOut	boolean	Flag indicating whether the object is to remain checked out after the call
remark	String	Character string that is added to the log of the WCM object or null
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

none

Prerequisites

- Required access rights: READ and WRITE
- Object status: must be CHECKED_OUT

Result

After being returned, the WCM object is in the status

- EDITED if keepCheckOut had the value false
- CHECKED_OUT if keepCheckOut had the value true

checkOut

Checks out the specified WCM object

Parameters

Table 30 – Parameters of the method `checkOut`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the respective WCM object
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `WRITE`
- Object status: must be `EDITED`, `REJECTED`, `RELEASED`, or `PENDING_RELEASE`

Result

After being checked out, the WCM object is in the `CHECKED_OUT` status.

> **Note:** To edit the object content, `getCheckOutContent` must be called afterwards, see “getCheckOutContent” on page 129.

directRelease

Releases a list of WCM objects without submitting them to Quality Assurance

Parameters

Table 31 – Parameters of the method directRelease

Parameter	Data type	Description
oids	ObjectId[]	OIDs of the WCM objects to be released directly
pendingReleaseDate	Date	Date and time when the specified WCM objects are to be released (delayed release)
remark	String	Character string that is added to the log of the WCM object or null
emailInfo	EMailInfo	E-mail information that is sent to the recipients defined in the object data, or null if no e-mail is to be sent
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

none

Prerequisites

- Required access rights: READ and WRITE or WRITE_META
- Object status: must be EDITED

Result

After direct release, the WCM object is in the status `RELEASED` or `PENDING_RELEASE` (if the release date lies in the future).

`reject`

Rejects a list of WCM objects

Parameters

Table 32 – Parameters of the method `reject`

Parameter	Data type	Description
`oids`	`ObjectId[]`	OIDs of the WCM objects to be rejected
`remark`	`String`	Character string that is added to the log of each WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `RELEASE`
- Object status: must be `SUBMITTED` or `DELETED`

Result

After having been rejected, the WCM objects are in the `REJECTED` status.

`release`

Releases a list of WCM objects

Parameters

Table 33 – Parameters of the method `release`

Parameter	Data type	Description
`oids`	`ObjectId[]`	OIDs of the WCM objects to be released
`remark`	`String`	Character string that is added to the log of each WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `RELEASE`
- Object status: must be `SUBMITTED`

Result

After having been released, the WCM objects are in the `RELEASED` or `PENDING_RELEASE` status. The status change at release depends on whether the “delayed release” metadata item is set.

`submit`

Submits a list of WCM objects to Quality Assurance

Parameters

Table 34 – Parameters of the method `submit`

Parameter	Data type	Description
`oids`	`ObjectId[]`	OIDs of the WCM objects to be submitted
`pendingReleaseDate`	`Date`	Date and time in the case of delayed release or `null` if immediate release is to be allowed
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `WRITE` or `WRITE_META`
- Object status: must be `EDITED` or `REJECTED`

Result

After having been submitted, the WCM objects are in the `SUBMITTED` status.

`submitImmediately`

Submits a list of WCM objects to Quality Assurance and enables direct release

Parameters

Table 35 – Parameters of the method `submitImmediately`

Parameter	Data type	Description
`oids`	`ObjectId[]`	OIDs of the WCM objects to be submitted
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: READ and WRITE or WRITE_META
- Object status: must be EDITED or REJECTED

Result

After having been submitted, the WCM objects are in the SUBMITTED status.

undoCheckOut

Undoes check out of specified WCM objects. This makes the objects available for others to edit.

Parameters

Table 36 – Parameters of the method undoCheckOut

Parameter	Data type	Description
oids	ObjectId[]	OIDs of the respective WCM objects
remark	String	Character string that is added to the log of the WCM object or null
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

none

Prerequisites

- Required access rights: READ and WRITE
- Object status: must be CHECKED_OUT

Result

After the changes to the checked out WCM objects have been discarded, the WCM objects are in the `EDITED` status.

Workflow Methods

The interface for managing workflow objects allows you to assign workflows to WCM objects, remove this assignment, and forward WCM objects in the workflow.

This section introduces the methods of WCM WebServices that can be used to perform workflow actions.

`assignWorkflow`

Assigns the currently released version of a workflow object to the specified WCM object. An initial transition must be selected. After the assignment, this transition is automatically executed.

Parameters

Table 37 – Parameters of the method `assignWorkflow`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the WCM object to which you want to assign a workflow
`workflowOid`	`ObjectId`	OID of the assigned workflow
`startTransition`	`String`	Name of the initial transition

Parameter	Data type	Description
remark	String	Character string that is added to the log of the WCM object or null
emailInfo	EMailInfo	The e-mail information that is sent to the principal assigned to the next activity, or null if no e-mail is to be sent.

Return

none

Prerequisites

- Required access rights: READ
- Object status: is determined by the initial workflow activity
- Functional area: WORKFLOW

Result

The object status does not change.

assignWorkflowAsync

Assigns the currently released version of a workflow object to **several** WCM objects. An initial transition must be selected. After the assignment, this transition is automatically executed. This method is asynchronous, i.e. it returns almost immediately after its call and does not wait for the related tasks to be finished.

Parameters

Table 38 – Parameters of the method `assignWorkflowAsync`

Parameter	Data type	Description
`oids`	`ObjectId[]`	OIDs of the WCM objects to which you want to assign a workflow
`workflowOid`	`ObjectId`	OID of the assigned workflow
`startTransition`	`String`	Name of the initial transition
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	The e-mail information that is sent to the principal assigned to the next activity, or `null` if no e-mail is to be sent.

Return

none

Prerequisites

- Required access rights: `READ`
- Object status: is determined by the initial workflow activity
- Functional area: `WORKFLOW`

Result

The object status does not change.

`forward`

Forwards an object in the workflow via the specified transition from the current activity to the next

Parameters

Table 39 – Parameters of the method `forward`

Parameter	Data type	Description
`workflow`	`Workflow`	Workflow definition
`transition`	`Transition`	Transition referencing the next activity
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	The e-mail information that is sent to the principal assigned to the next activity, or `null` if no e-mail is to be sent.

Return

none

Prerequisites

The required access rights and functional areas are determined by the workflow activity.

Result

The object status is determined by the last workflow activity.

`getAssignedJobs`

Returns the IDs of all WCM objects that are assigned a certain workflow and that are in an activity to which the logged-in principal is assigned.

Parameters

Table 40 – Parameters of the method `getAssignedJobs`

Parameter	Data type	Description
`workflow`	`Workflow`	Workflow definition or `null`
`activityId`	`String`	Activity ID or `null`

Return

`ObjectId[] objectIds`: a list of OIDs

Prerequisites

none

Result

The object status does not change.

`getAssignedWorkflow`

Returns the assigned workflow instance for the specified WCM object

Table 41 – Parameters of the method `getAssignedWorkflow`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the WCM object for which you want to determine the assigned workflow

Return

`Workflow workflow`: the currently assigned workflow object

Prerequisites

None

Result

The object status does not change.

getAssignedWorkflowActivities

Returns a list of `WorkflowNavigationInfo` objects whose activities can be performed by the logged-in principal

Return

`WorkflowNavigationInfo[] workflowNavigationInfos`: a set of `WorkflowNavigationInfo` objects

Prerequisites

None

Result

The object status does not change.

getAssignedWorkflows

Returns all workflow objects whose current activity is assigned to the logged-in principal

Return

`Workflow workflows`: a set of workflow objects

Prerequisites

None

Result

The object status does not change.

getAvailableWorkflows

Returns all workflow objects for which the principal has the access right READ

Return

Workflow workflows: a set of workflow objects

Prerequisites

- Required access right: READ
- Object status: no restrictions

Result

The object status does not change.

getWorkflow

Returns the workflow definition in the version specified

Table 42 – Parameters of the method getWorkflow

Parameter	Data type	Description
workflowOid	ObjectId	OID of the workflow object
version	Version	Version of the workflow object

Return

Workflow workflow: a workflow object

Prerequisites

- Required access right: READ
- Object status: no restrictions

Result

The object status does not change.

isForwardable

Checks whether the logged-in principal is assigned to the current activity of the WCM object

Table 43 – Parameter of the method `isForwardable`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the WCM object a workflow is assigned to

Return

`true`, if the principal is allowed to forward the object in the workflow, `false` otherwise

Prerequisites

None

Result

The object status does not change.

isRemovable

Checks whether the logged-in principal is allowed to remove the assignment between workflow object and WCM object

Table 44 – Parameter of the method `isRemovable`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the workflow object

Return

`true`, if the principal is allowed to remove the assignment between workflow object and WCM object, `false` otherwise

Prerequisites

None

Result

The object status does not change.

removeWorkflow

Removes the assignment between workflow object and WCM object

Table 45 – Parameters of the method removeWorkflow

Parameter	Data type	Description
oid	ObjectId	OID of the WCM object whose assignment to a workflow object is to be removed
remark	String	Character string that is added to the log of the WCM object or null

Return

None

Prerequisites

- Required access right: READ and CHANGE_RIGHTS
- Object status: no restrictions

Result

The object status does not change.

removeWorkflowAsync

Removes the assignment between workflow object and **several** WCM objects. This method is asynchronous, i.e. it returns almost immediately after its call and does not wait for the related tasks to be finished.

Table 46 – Parameters of the method removeWorkflowAsync

Parameter	Data type	Description
oids	ObjectId[]	OIDs of the WCM objects whose assignment to a workflow object is to be removed
remark	String	Character string that is added to the log of the WCM object or null

Return

None

Prerequisites

- Required access right: READ and CHANGE_RIGHTS
- Object status: no restrictions

Result

The object status does not change.

Object Management Methods

This section introduces the methods of WCM WebServices that can be used to edit WCM objects.

Adding objects

- creating new objects, see “create” on page 122
- creating objects by means of an import, see “multiImport” on page 137

Retrieving and editing objects

- reading object with the specified OID in the defined version, see “get” on page 128
- receiving content of the WCM object with the specified OID and the defined version. The content is generated by the assigned deployment system. In this process, the content is merged with the templates used. See “getCheckOutContent” on page 129.
- receiving content of the WCM object with the specified OID and the defined version without further manipulation of the content with regard to the template used. See “getContent” on page 131.
- changing metadata, see “change” on page 117
- converting compound objects, see “convertContent” on page 120

Retrieving/sorting position in the hierarchy

- retrieving the subordinate objects of a topic, see “getChildren” on page 130
- retrieving the parent object, see “getParent” on page 134
- sorting elements in such a way that each child object is placed behind its parent object, see “sortParentsFirst” on page 140

Copy and move objects

- copying objects, see “copy” on page 121
- moving object, see “move” on page 136

Deleting objects

- deleting objects (must be confirmed by Quality Assurance), see “delete” on page 124
- deleting objects finally, see “destroy” on page 126

Log and versions

- requesting log entries, see “getLastLogEntries” on page 133
- adding remark to log, see “addRemark” on page 116
- listing versions of an object, see “getVersionList” on page 135
- restoring a previous object version, see “restoreVersion” on page 139

Checking references

- checking before deletion whether the specified WCM objects are referenced by other WCM objects, see “checkReferencesForDelete” on page 118
- checking before release whether the specified WCM objects reference other WCM objects that have not been released yet, see “checkReferencesForRelease” on page 118
- checking before submitting whether the specified WCM objects reference other WCM objects that have not been submitted to Quality Assurance yet, see “checkReferencesForSubmit” on page 119
- requesting external references, see “getExternalLinks” on page 132

Deployment

- generating page(s), see “generatePage” on page 127
- deleting generated page(s) from the file system, see “depublishPage” on page 125
- requesting remaining deployment jobs for a specified deployment system and a WCM object, see “getDeploymentJobs” on page 132

In the following overview, the methods are sorted alphabetically.

addRemark

Adds a remark to the log of the specified WCM object.

Parameters

Table 47 – Parameters of the method `addRemark`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the respective WCM object
`remark`	`String`	Character string that is added to the log of the WCM object or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `WRITE_META`
- Object status: no restrictions

Result

The object status does not change.

change

Changes the metadata of a WCM object

Only the attributes indicated as modifiable in table “The components of the data type `ObjectData`” on page 75 can be changed.

In `objectData.attributeKeys`, all attributes to be changed must be specified. Moreover, `objectData.objectId` must be set.

Parameters

Table 48 – Parameters of the method change

Parameter	Data type	Description
`objectData`	`ObjectData`	Changed object data
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `WRITE_META`. If the access control list (ACL) is to be changed, the access right `CHANGE_RIGHTS` is also required.
- Object status: must be `EDITED`, `REJECTED`, `RELEASED`, or `PENDING_RELEASE`

Result

After the change, the specified WCM object is in the `EDITED` state.

checkReferencesForDelete

Method for checking whether the specified WCM objects are referenced by other WCM objects. This method should be called before deleting or destroying WCM objects.

Parameters

Table 49 – Parameter of the method `checkReferencesForDelete`

Parameter	Data type	Description
`oids`	`ObjectId[]`	List of OIDs whose associated WCM objects are to be checked

Return

`ObjectId[] objectIds`: A list of OIDs which are contained in `oids` and whose associated WCM objects are referenced by other WCM objects.

Prerequisites

- No access rights are required
- Object status: no restrictions

Result

The object status does not change.

checkReferencesForRelease

Method for checking whether the specified WCM objects reference other WCM objects that have not been released yet. This method should be called before releasing WCM objects.

Parameters

Table 50 – Parameter of the method `checkReferencesForRelease`

Parameter	Data type	Description
`oids`	`ObjectId[]`	List of OIDs whose associated WCM objects are to be checked

Return

`ObjectId[] objectIds`: A list of OIDs that are contained in `oids` and whose associated WCM objects reference other WCM objects that are not contained in `oids` and that have not been released yet. Only WCM objects that are available in the QA view and that have thus been submitted to Quality Assurance at least once are returned.

Prerequisites

- No access rights are required.
- Object status: no restrictions

Result

The object status does not change.

`checkReferencesForSubmit`

Method for checking whether the specified WCM objects reference other WCM objects that have not been submitted to Quality Assurance yet. This method should be called before submitting WCM objects to Quality Assurance.

Parameters

Table 51 – Parameter of the method `checkReferencesForSubmit`

Parameter	Data type	Description
`oids`	`ObjectId[]`	List of OIDs whose associated WCM objects are to be checked

Return

`ObjectId[] objectIds`: a list of OIDs that are contained in `oids` and whose associated WCM objects reference other WCM objects that are not contained in `oids` and that have not been submitted to Quality Assurance yet.

Prerequisites

- No access rights are required.
- Object status: no restrictions

Result

The object status does not change.

`convertContent`

Converts the content of a compound object. The subordinate objects of a compound object are created.

Parameters

Table 52 – Parameters of the method `convertContent`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the compound object
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

None

Prerequisites

- Required access right for the source object: `READ`
- Required access rights for the topic containing the source object: `READ`, `CREATE`, `DELETE`, `WRITE` and `WRITE_META`
- Object status of the source object: must be `EDITED`, `REJECTED`, `RELEASED`, or `PENDING_RELEASE`

Result

The object status of the compound object does not change. The object status of the subordinate objects generated during conversion is `EDITED`.

copy

Copies a WCM object to a location below a topic

The specified object may also be copied to a location below the topic to which it already belongs. This does not change the title of the object. In this case, two objects with the same name exist.

Parameters

Table 53 – Parameters of the method copy

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the respective WCM object
`newParentOID`	`ObjectId`	The OID of the topic into which the WCM object is to be copied.

Parameter	Data type	Description
remark	String	Character string that is added to the log of the WCM object or null
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

ObjectId objectId : the OID of the new, copied WCM object

Prerequisites

- Required access rights for the copied WCM object: READ, TREE_OPERATIONS, and WRITE_META
- Required access rights for the target topic: READ and CREATE
- Object status: must be EDITED, REJECTED, RELEASED, or PENDING_RELEASE

Result

The status of the copied WCM object (the source object) does not change. The newly created copy acquires the status EDITED.

create

Creates a new WCM object with the specified metadata. The template in the object data (objectData.template) may be null.

Parameters

Table 54 – Parameters of the method `create`

Parameter	Data type	Description
`objectData`	`ObjectData`	For a new object, at least the attributes `topic`, `objectType`, and `title` must be specified. If the new object is to be based on a template, `template` must also be specified. Access rights (ACL) cannot be defined here.
`content`	`byte[]`	Content of the object or `null` of no content is to be specified.
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

`ObjectId objectId` : the OID of the newly created WCM object

Prerequisites

- Required access rights for the parent topic: `READ` and `CREATE`
- Object status: no restrictions
- The required functional areas depend on the object type of the new object.

Result

After having been created, the WCM object is in the status `EDITED`.

delete

Deletes the specified WCM objects

If a WCM object has never been submitted to Quality Assurance, this method completely removes it from the WCM system. Otherwise, it acquires the object status `DELETED` and remains visible in the QA view. There the deletion must be confirmed by means of a `destroy` action or rejected by means of `reject`.

Parameters

Table 55 – Parameters of the method `delete`

Parameter	Data type	Description
`oids`	`ObjectId[]`	A list of OIDs
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `DELETE`
- Object status: must be `EDITED`, `REJECTED`, `RELEASED`, or `PENDING_RELEASE`

Result

If a WCM object has already been submitted to Quality Assurance once, it acquires the status `DELETED` after deletion.

`depublishPage`

Deletes the pages generated for the specified WCM objects from the file system. The pages are thus removed from the Production view. The deleted pages will not be generated anew until the respective WCM objects are released again.

Parameters

Table 56 – Parameters of the method `depublishPage`

Parameter	Data type	Description
`dplSystem`	`String`	Name of a deployment system
`oids`	`ObjectId[]`	A list of OIDs
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access right: `READ` and `RELEASE`
- Object status: no restrictions

Result

The object status does not change.

destroy

Destroys the specified WCM objects and thus confirms the deletion performed by means of `delete`

Parameters

Table 57 – Parameters of the method `destroy`

Parameter	Data type	Description
`oids`	`ObjectId[]`	A list of OIDs
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`emailInfo`	`EMailInfo`	E-mail information that is sent to the recipients defined in the object data, or `null` if no e-mail is to be sent
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: `READ` and `RELEASE`
- Object status: must be `DELETED`

Result

After having been destroyed, the WCM object – including all old versions – is no longer visible in the WCM system.

generatePage

Generates the pages for the specified list of WCM objects (page generation). WCM objects whose pages have been removed from the Production view by means of the function *Remove page* cannot be regenerated by means of this method. Such objects have to go through the entire staging before they can be published again.

Parameters

Table 58 – Parameters of the method generatePage

Parameter	Data type	Description
dplSystem	String	Name of a deployment system
oids	ObjectId[]	A list of OIDs
useReleasedTemplate	boolean	true if the released template is to be used, false for the current version of the template
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

none

Prerequisites

- Required access right: READ
- Object status: no restrictions

Result

The object status does not change.

get

Returns the WCM object with the specified OID in the defined version. The parameter `attrKeys` can be used to request an exact list of attributes that are available in the return value after calling the method.

Parameters

Table 59 – Parameters of the method `get`

Parameter	Data type	Description
`oid`	`ObjectId`	The OID of the WCM object
`version`	`Version`	The desired version The number for the current version cannot be specified here, `null` returns the current version.
`attributeKeys`	`String[]`	A list of the desired attributes

Return

`ObjectData objectData`: the requested data of the specified WCM object

Prerequisites

- Required access rights: `READ` or `READ_PRODUCTION`
- Object status: no restrictions

Result

The object status does not change.

getCheckOutContent

Returns the content of the WCM object with the specified OID and the defined version in a byte array. This method can also be used if the WCM object is not checked out, i.e. if the state of the WCM object is not CHECKED_OUT.

The content is generated by the deployment system that is assigned to WCM WebServices. In the case of HTML objects and similar object types, the head elements of the templates are merged and inserted in the head element of the content. The body element is not changed, except for the contained references, which are updated, if necessary. WCM tags are not replaced. In the case of topics, the head elements are not merged.

In the case of binary WCM objects (e.g. GIF images), the content is not changed.

Parameters

Table 60 – Parameters of the method getCheckOutContent

Parameter	Data type	Description
oid	ObjectId	The OID of the WCM object
version	Version	The desired version, null returns the current version

Return

byte[] content: the checked out content of the requested WCM object

Prerequisites

- Required access right: READ
- Object status: no restrictions

Result

The object status does not change.

getChildren

Returns the OIDs of the child objects of the respective WCM object under the specified `view`

Parameters

Table 61 – Parameters of the method `getChildren`

Parameter	Data type	Description
`oid`	`ObjectId`	The OID of the WCM object
`view`	`int`	`1` = Topic tree view `2` = Template view `3` = Compound object view

Return

`ObjectId[] objectIds`: a list of OIDs of the requested child objects

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

getContent

Returns the content of the WCM object with the specified OID and the defined version in a byte array. This method returns the content of the WCM object as it is saved in the content repository without further manipulation of the content regarding the template used.

> **Note:** If the WCM object does not have any content, an exception is thrown if this method is called.

Parameters

Table 62 – Parameters of the method `getContent`

Parameter	Data type	Description
`oid`	`ObjectId`	The OID of the WCM object
`version`	`Version`	The desired version, `null` returns the content of the current version

Return

`byte[] content`: the content of the requested WCM object

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

getDeploymentJobs

Returns all remaining deployment jobs for a specified deployment system and a WCM object. Each staging action that changes a WCM object generates deployment jobs for the deployment system of a website.

Parameters

Table 63 – Parameters of the method `getDeploymentJobs`

Parameter	Data type	Description
`dplSystem`	`String`	Name of the deployment system
`oid`	`ObjectId`	OID of the respective WCM object

Return

`DeploymentJobInfo[] deplJobs`: the list of remaining deployment jobs. If there are no jobs for the specified parameters, an empty list is returned.

Prerequisites

- No access rights are required.
- Object status: no restrictions

Result

The object status does not change.

getExternalLinks

Returns all external references contained in the list of the specified WCM objects. External references point to URLs outside the WCM-managed website.

Parameters

Table 64 – Parameter of the method `getExternalLinks`

Parameter	Data type	Description
`oids`	`ObjectId[]`	The OIDs of WCM objects to be considered

Return

`LinkInfo[] links`: the list of external links

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

`getLastLogEntries`

Returns a list of log entries for the specified WCM object. The log entries are returned in descending order, starting with `first`.

Parameters

Table 65 – Parameters of the method `getLastLogEntries`

Parameter	Data type	Description
`oid`	`ObjectId`	The OID of the WCM object
`first`	`int`	The number of the log entry to be returned first; -1 starts with the last log entry
`number`	`int`	The number of log entries to be returned; -1 means all entries until the first

Return

`LogEntry[] logEntries`: the list of requested log entries

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

`getParent`

Returns the OIDs of the WCM objects that are located above the specified WCM object in the specified `view` (“parent objects”)

Parameters

Table 66 – Parameters of the method `getParent`

Parameter	Data type	Description
`oid`	`ObjectId`	The OID of the WCM object
`view`	`int`	`1` = Topic tree view `2` = Template view `3` = Compound object view
`toRoot`	`boolean`	`true` if all parent objects up to the root are to be returned

Return

`ObjectId[] objectIds`: a list of OIDs of the requested parent objects. If `toRoot` had the value `false`, exactly one object is returned. For the root object, an empty list is returned.

Prerequisites

- No access rights are required.
- Object status: no restrictions

Result

The object status does not change.

`getVersionList`

Returns the complete list of all versions of the specified WCM object.

Parameters

Table 67 – Parameter of the method `getVersionList`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the respective WCM object

Return

`Version[] versions`: the requested list of all versions for the specified WCM object

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

move

Moves the specified WCM object to a location below a different topic

Parameters

Table 68 – Parameters of the method move

Parameter	Data type	Description
oid	ObjectId	OID of the respective WCM object
newParentOID	ObjectId	The OID of the topic to which the WCM object is to be moved.
remark	String	Character string that is added to the log of the WCM object or null
dplWaitInfo	DeploymentWaitInfo	List of the deployment systems or null

Return

none

Prerequisites

- Required access rights for the moved WCM object: READ, TREE_OPERATIONS, and WRITE_META
- Required access rights for the target topic: READ and CREATE
- Object status: must be EDITED, RELEASED, or PENDING_RELEASE

Result

After having been moved, the specified WCM object is in the EDITED state.

multiImport

Generates several new WCM objects in one call. First, the initial object (`imports.objectData`) is created. After this, the objects defined in `imports.children[]` are created. The import differentiates the following possibilities:

- If the initial object has content (`imports.content != null`), this object is interpreted as the start page for importing linked objects. Of the objects specified in `imports.children[]`, only the objects referenced directly or indirectly by this WCM object are created.
- If the initial object does not have any content, all objects supplied are created.

When these individual objects are created, explicit hyperlinks to the existing website and relative hyperlinks to the imported objects are recognized and integrated into the reference management of Livelink WCM Server.

Parameters

Table 69 – Parameters of the method `multiImport`

Parameter	Data type	Description
`initialData`	`ObjectData`	Here, the topic (`topic`), object type (`objectType`), suggested file name (`deploymentHint`), and the title (`title`) for the object to be created are specified. All other attributes are adopted automatically for the subordinate objects and overwrite the values specified in `imports.children[].objectData`. This makes it possible to define uniform attributes for all objects. Attributes that are only valid for the start object of the import should be determined in `imports.children[0].objectData`. The import generates a separate topic if `initialData.objectType` is of type “Topic”. Otherwise, the objects to be generated will be created below the topic `initialData.topic`.
`imports`	`MultiImportPart`	Metadata, content, and subordinate objects See “Data Type: `MultiImportPart`” on page 88
`remark`	`String`	A character string that is added to the log of all WCM objects created or `null`
`dplWaitInf`	`DeploymentWaitAbout`	List of the deployment systems or `null`

Return

`ObjectId objectId` : the OID of the start object for the import

Prerequisites

- Required access rights for the parent topic: `READ` and `CREATE`
- Object status: no restrictions
- The required functional areas depend on the created object types.

Result

After the import, all created WCM objects are in the `EDITED` state.

`restoreVersion`

Restores an older version of the specified WCM object.

Parameters

Table 70 – Parameters of the method `restoreVersion`

Parameter	Data type	Description
`oid`	`ObjectId`	OID of the respective WCM object
`version`	`Version`	The desired version of the WCM object
`remark`	`String`	Character string that is added to the log of the WCM object or `null`
`dplWaitInfo`	`DeploymentWaitInfo`	List of the deployment systems or `null`

Return

none

Prerequisites

- Required access rights: READ, WRITE, and WRITE_META
- Object status: must be EDITED, REJECTED, RELEASED, or PENDING_RELEASE

Result

After the specified version has been restored, the WCM object is in the EDITED state.

sortParentsFirst

Returns a list in which each child object is placed directly after its direct parent object according to the specified view. This method is useful for sorting WCM objects that are to be submitted to Quality Assurance or released.

Parameters

Table 71 – Parameters of the method SortParentsFirst

Parameter	Data type	Description
oids	ObjectId[]	The OIDs of the WCM objects to be sorted
topologyView	int	1 = Topic tree view 2 = Template view 3 = Compound object view

Return

ObjectId[] objectIds: a list containing the newly sorted OIDs

Prerequisites

- No access rights are required
- Object status: no restrictions

Result

The object status does not change.

Searching for WCM Objects

The `filter` method is used to search for WCM objects. A search condition is represented as an object of type `Filter`. The data types based on the type `Filter` can be used to model logical expressions. Complex filters can be compiled by means of links.

There are different categories of filter types:

- filters that are based on the attributes of WCM objects (for information on searchable attributes, see table "The components of the data type `ObjectData`" on page 75). These filters are of type `IsNullFilter`, `JoinFilter`, or extend the data type `ValueFilter`.
- filters that logically link existing filters and thus allow the construction of complex queries. The following filter types belong to this category: `NotFilter`, `AndFilter`, and `OrFilter`.
- filters that execute a predefined search function. The following filter types belong to this category: `RootTemplateFilter`, `PermissionFilter`, `SubtreeFilter`, and `PrincipalFilter`.

The following overview illustrates the data type `Filter` and the derived filter types.

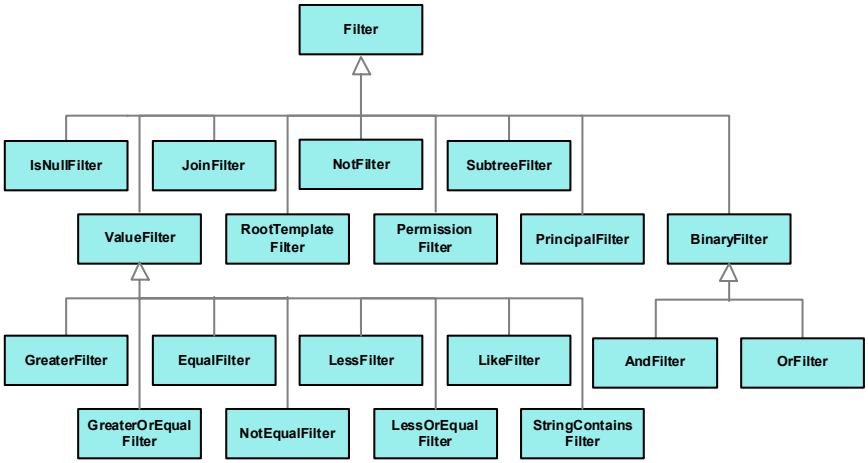


Fig. 7 – The data type `Filter` and derived special filter types

Attribute Value Filters (`ValueFilter`)

Livelink WCM Server manages the data of a website (including the content and the metadata) in a database. When applying the search function, a query suited for the respective database system (a SQL command in the form `SELECT ... FROM ... WHERE ... ORDER BY ...`) is formulated on the basis of the filter object. The attribute names correspond to the components of the data type `ObjectData`.

However, there are some attributes which cannot be used for formulating SQL queries. For this reason, not all attributes can be used in a search procedure. For detailed information about searchable attributes, refer to table “The components of the data type `ObjectData`” on page 75.

The following table can be used to determine which attribute types can be used together with which filter types.

Table 72 – Attribute types and filters

	LikeFilter	StringContainsFilter	IsNullFilter[a]	JoinFilter[b]	EqualFilter	NotEqualFilter	GreaterFilter	GreaterOrEqualFilter	LessFilter	LessOrEqualFilter
IntegerValue, LongValue			✓	✓	✓	✓	✓	✓	✓	✓
ListValue[c], SetValue[c]	✓	✓	✓							
DateValue			✓	✓	✓	✓	✓	✓	✓	✓
BooleanValue			✓	✓	✓	✓				
StringValue	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LocaleValue[d], ObjectId, ObjectType, ObjectState, User[d], Version				✓ ✓	✓					

(a) This filter can be used to check whether a value has been set for an attribute.

(b) This filter compares two attribute values. Both values must be of the same type.

(c) A list entry is searched which is specified as a `StringValue`.

(d) In the case of `LocaleValue` and `User`, the value can also be specified as a `StringValue`; so the other filters can be used as well.

Examples

```
new IsNullFilter("subtitle")

new StringContainsFilter("QAEMailReceivers",
      new StringValue("jstein@company.example"))

new LikeFilter("QAEMail*",
      new StringValue("jstein@*"))

new EqualFilter("createdBy", new StringValue("jstein"))
```

Predefined Search Functions as Filters

- The filter type `RootTemplateFilter` searches for all WCM objects of a website that are of the object type “Template” and that are not assigned to a template themselves.
- The filter type `PermissionFilter` searches for all WCM objects that have been assigned a given user with a given right in the access control list.
- Using the filter type `SubtreeFilter`, all child objects of a WCM object (topic in the topic structure or template in the template structure) can be determined. The `SubtreeFilter` can also be used without entering the `Topology` parameter. In this case, the topic structure is considered to be a default value.
- The filter type `PrincipalFilter` searches for WCM objects that have been assigned a certain principal (user, group, or role). In addition, it is possible to limit the result to WCM objects for which the principal has a certain right (`Permission`). For further information on principals, refer to section “Users, Groups, and Roles” on page 52.

Examples

The following filter searches for all WCM objects that the user “jstein” is allowed to read.

```
new PermissionFilter(new Permission("r"), new User("jstein") )
```

The following filter returns all WCM objects (including the specified start node) that are located below the WCM object with the OID “4711” in the topic structure.

```
new SubtreeFilter(new ObjectId("4711") )
```

The Method `filter`

This method finds objects below the specified topic that match the defined filter criterion.

The result can be sorted by specifying a list of sort objects which determines the attributes to be sorted and the sort order. Moreover, it is possible to use special attributes for constructing a search filter.

> **Note:** At present, the `filter` method does not yet work properly when specifying special attributes for sorting the hitlist.

WCM objects for which the user of the method does not have read rights are automatically excluded from the search.

The WCM object defined as the starting point of the search must be a topic.

The result set can be restricted by `startResult` and `numberOfResults`.

Parameters

Table 73 – Parameters of the method `filter`

Parameter	Data type	Description
`filter`	`Filter`	The search criterion or `null` to search for all objects
`startOID`	`ObjectId`	By entering the OID of a start node, the search starts below the specified WCM object. In this case, the start node itself (`startOID`) and all child objects in the topic structure are taken into consideration. The `startOID` must reference a WCM object of the type “Topic”.
`sortList`	`Sort[]`	A list of attribute constants according to which the result set is to be sorted. By default, the hitlist is sorted by OID.
`startResult`	`int`	A number specifying the first element of the search result (usually 0)
`numberOfResults`	`int`	Number of search results to be returned, starting at `startResult` -1 for all results
`attributeKeys`	`String[]`	The names of all attributes that are to be set in the returned object data

Return

`ObjectData[] objectData`: a list of object data, i.e. the data of the WCM objects matching the specified search criterion.

Prerequisites

- Required access right: `READ`
- Object status: no restrictions

Result

The object status does not change.

CHAPTER 6
Application Examples

One of the main purposes of the Web Service technology is to easily find and use a Web Service. UDDI is responsible for finding the Web Services. This topic is not described in this manual. As Web Services describe themselves by WSDL, many manufacturers have provided tools that read a Web Services definition and automatically convert it into a module that can be used by a client application. As an application developer, you integrate this module in your development environment and write your application in a programming language supported by the tool.

This chapter contains two examples for using WCM WebServices. Of the wide variety of programming languages that can use Web Services, VisualBasic for Applications (VBA) – as an example of a non-object-oriented script language – and C# in connection with VisualStudio.Net will be used as examples.

This chapter does not intend to offer a programming course in these languages, it will draw your attention to the special aspects of working with these languages and will present prototypical code.

The examples refer to the fictional website “InternetSite” of the company “company.example”:

- Example 1 demonstrates how to formulate complex filter expressions and how to edit the result sets.
- Example 2 shows how to use the staging functions of Livelink WCM Server.

In the context of these examples, you also learn how authentication, session management, and error handling is realized in the two programming environments.

VisualBasic for Applications

VisualBasic for Applications is used for integrating new functions in Microsoft Office applications. For using Web Services, Microsoft provides the so-called “SOAP toolkit”. This toolkit must be installed on the client computers. The SOAP toolkit is responsible for correctly sending and receiving the SOAP messages. You as the application developer must generate the XML content. Thus, the techniques described in this chapter can also be applied to other script languages.

Structure of a Web Service Call

At first, the structure of a Web Service call will be explained:

Each call begins with the SOAP envelope followed by the SOAP body. The SOAP body contains exactly one element represented by the name of the method to be called. This element contains the parameters of the method. Empty parameters may be left out.

The following example shows a complete request (including HTTP header) as it is generated in the Microsoft .NET Framework SDK for the following method call:

```
filter('objectType == PDF', ...)
```

```
POST /wcm/WebService/Port/InternetSite/edit HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client
Protocol 1.0.3705.0)
Content-Type: text/xml;
charset=utf-8
SOAPAction: "http://gaussvip.com/"
```

```
Authorization: Basic YWRtaW46YQ==
Content-Length: 1820
Expect: 100-continue
Host: wcmserver.company.example
Cookie: JSESSIONID=B8D157BC6E214885EE2A2A3A0FA788C8

<?xml version="1.0" encoding="utf-8"?>
 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:tns="http://gaussvip.com/"
                xmlns:types="http://gaussvip.com/encodedTypes"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body
         soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <tns:filter>
    <filter href="#id1" />
    <startOID href="#id2" />
    <sortList href="#id3" />
    <startResult     xsi:type="xsd:int">0</startResult>
    <numberOfResults xsi:type="xsd:int">5</numberOfResults>
    <attributeKeys href="#id4" />
   </tns:filter>
   <tns:EqualFilter id="id1" xsi:type="tns:EqualFilter">
    <key href="#id5" />
    <value href="#id6" />
    <caseSensitive xsi:type="xsd:boolean">false</caseSensitive>
   </tns:EqualFilter>
   <tns:ObjectId id="id2" xsi:type="tns:ObjectId">
    <id xsi:type="xsd:string">1 </id>
   </tns:ObjectId>
   <soapenc:Array id="id3" soapenc:arrayType="tns:Sort[1]">
    <Item href="#id7" />
   </soapenc:Array>
   <soapenc:Array id="id4" soapenc:arrayType="xsd:string[4]">
    <Item>createdDate</Item>
    <Item>objectId   </Item>
    <Item>objectState</Item>
    <Item>title      </Item>
   </soapenc:Array>
   <tns:Key id="id5" xsi:type="tns:Key">
    <stringValue xsi:type="xsd:string">objectType</stringValue>
   </tns:Key>
   <tns:ObjectType id="id6" xsi:type="tns:ObjectType">
    <name xsi:type="xsd:string">PDF</name>
    <template xsi:type="xsd:boolean">false</template>
    <topic xsi:type="xsd:boolean">false   </topic>
    <frame xsi:type="xsd:boolean">false   </frame>
    <fileOnCreateNeeded xsi:type="xsd:boolean">false
```

```
    </fileOnCreateNeeded>
   </tns:ObjectType>
   <tns:Sort id="id7" xsi:type="tns:Sort">
    <stringValue xsi:type="xsd:string">createdDate</stringValue>
    <descending xsi:type="xsd:boolean">false      </descending>
   </tns:Sort>
  </soap:Body>
 </soap:Envelope>
```

This example is very complex. The reason lies in the general approach that the .NET Framework SDK (like other toolkits) uses for serializing objects in XML: it tries to serialize equal objects only once. This is not necessarily required. Thus, an equivalent SOAP command can look that simple:

```
POST /wcm/WebService/Port/InternetSite/edit HTTP/1.1
Authorization: Basic YWRtaW46YQ==
Content-Type: text/xml
Host: wcmserver.company.example
SOAPAction: "http://gaussvip.com/
"Content-Length: 1378

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:vip="http://gaussvip.com/">
  <SOAP-ENV:Body>
   <SOAPSDK1:filter xmlns:SOAPSDK1="http://gaussvip.com/">
    <filter xsi:type="vip:EqualFilter">
     <key xsi:type="vip:Key">
      <stringValue xsi:type="xsd:string">objectType</stringValue>
     </key>
     <value xsi:type="vip:ObjectType">
      <name type="xsd:string">PDF</name>
     </value>
    </filter>
    <startOID xsi:type="vip:ObjectId">
     <id xsi:type="xsd:string">1 </id>
    </startOID>
    <sortList
            xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="vip:Sort[1]">
```

```
     <Item xsi:type="vip:Sort">
      <stringValue xsi:type="xsd:string">createdDate</stringValue>
      <descending  xsi:type="xsd:boolean">False      </descending>
     </Item>
    </sortList>
    <startResult     xsi:type="xsd:int">0</startResult>
    <numberOfResults xsi:type="xsd:int">5</numberOfResults>
    <attributeKeys
          xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[4]">
     <Item xsi:type="xsd:string">createdDate  </Item>
     <Item xsi:type="xsd:string">objectId     </Item>
     <Item xsi:type="xsd:string">objectState  </Item>
     <Item xsi:type="xsd:string">title        </Item>
    </attributeKeys>
   </SOAPSDK1:filter>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

The following sections show how to construct such simply formulated SOAP commands. The response received by the server – also in the form of a SOAP envelope – is at least as complex as the above request. However, this complexity cannot be controlled. Thus, the more complex part of the programming example deals with the deserialization of the data received.

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Content-Type: text/xml;
charset=utf-8
Content-Length: 4151
Date: Thu, 18 Apr 2002 07:30:32 GMT
Server: Apache Tomcat/4.0.3 (HTTP/1.1 Connector)

<?xml version="1.0" encoding="UTF-8"?>
 <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:filterResponse
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns:ns1="http://gaussvip.com/">
```

```
 <filterResult xsi:type="SOAP-ENC:Array"
       SOAP-ENC:arrayType="ns1:ObjectData[5]"
       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
  <item href="#id1"/>
  <item href="#id2"/>
  <item href="#id3"/>
  <item href="#id4"/>
  <item href="#id5"/>
 </filterResult>
 </ns1:filterResponse>
 <multiRef id="id5" SOAP-ENC:root="0" xsi:type="ns3:ObjectData"
      xmlns:ns3="http://gaussvip.com/"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">

 <createdDate xsi:type="xsd:dateTime">2002-04-17T16:45:32.000Z
 </createdDate>
 <objectId href="#id6"/>
 <objectState href="#id7"/>
 <title xsi:type="xsd:string">8.0.1_portalManagerProgrammersManual_en
 </title>
</multiRef>
<multiRef id="id2" SOAP-ENC:root="0" xsi:type="ns4:ObjectData"
          xmlns:ns4="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <createdDate xsi:type="xsd:dateTime">2002-04-17T16:45:08.000Z
 </createdDate>

 <objectId href="#id8"/>
 <objectState href="#id7"/>
 <title
       xsi:type="xsd:string">8.0.1_contentManagerProgrammersManual_en
 </title>
</multiRef>
<multiRef id="id4" SOAP-ENC:root="0" xsi:type="ns5:ObjectData"
          xmlns:ns5="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <createdDate xsi:type="xsd:dateTime">2002-04-17T16:45:24.000Z
 </createdDate>
 <objectId href="#id9"/>
 <objectState href="#id7"/>
 <title xsi:type="xsd:string">8.0.1_contentMinerManual_en
 </title>
</multiRef>
<multiRef id="id3" SOAP-ENC:root="0" xsi:type="ns6:ObjectData"
          xmlns:ns6="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <createdDate xsi:type="xsd:dateTime">2002-04-17T16:45:18.000Z
 </createdDate>
 <objectId href="#id10"/>
```

```
 <objectState href="#id7"/>
 <title xsi:type="xsd:string">8.0.1_contentManagerUserManual_en
 </title>
</multiRef>
<multiRef id="id1" SOAP-ENC:root="0" xsi:type="ns7:ObjectData"
          xmlns:ns7="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <createdDate xsi:type="xsd:dateTime">2002-04-17T16:45:06.000Z
 </createdDate>
 <objectId href="#id11"/>
 <objectState href="#id7"/>
 <title xsi:type="xsd:string">VipNote20_PomaMigration_en
 </title>
</multiRef>
<multiRef id="id6" SOAP-ENC:root="0" xsi:type="ns8:ObjectId"
          xmlns:ns8="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <id xsi:type="xsd:string">62 </id>
</multiRef>

<multiRef id="id9" SOAP-ENC:root="0" xsi:type="ns9:ObjectId"
          xmlns:ns9="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <id xsi:type="xsd:string">61 </id>
</multiRef>
<multiRef id="id10" SOAP-ENC:root="0" xsi:type="ns10:ObjectId"
          xmlns:ns10="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <id xsi:type="xsd:string">60 </id>
</multiRef>
<multiRef id="id8" SOAP-ENC:root="0" xsi:type="ns11:ObjectId"
          xmlns:ns11="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <id xsi:type="xsd:string">59 </id>
</multiRef>
<multiRef id="id7" SOAP-ENC:root="0" xsi:type="ns12:ObjectState"
          xmlns:ns12="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <name xsi:type="xsd:string">edited </name>
 <description xsi:type="xsd:string">ge&#228;ndert</description>
 <imageURL xsi:type="xsd:string">
    http://wcmserver.company.example/wcm/InternetSite_edit/
                                  vipimages/objectstate/changed.gif
 </imageURL>
</multiRef>
<multiRef id="id11" SOAP-ENC:root="0" xsi:type="ns13:ObjectId"
          xmlns:ns13="http://gaussvip.com/"
          xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
 <id xsi:type="xsd:string">58 </id>
```

```
  </multiRef>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Preparations

The SOAP toolkit is available for download at http://download.microsoft.com/download/xml/soap/2.0/W98NT42KMe/EN-US/SoapToolkit20.exe. It contains version 3 (MSXML3) of the Microsoft XML parser. For example, if you want to create an add-in for Microsoft Word, create a document template in Word and open the VisualBasic editor by pressing ALT+ F11. The SOAP toolkit and the XML parser must be entered as references (menu *Tools* → *References*):

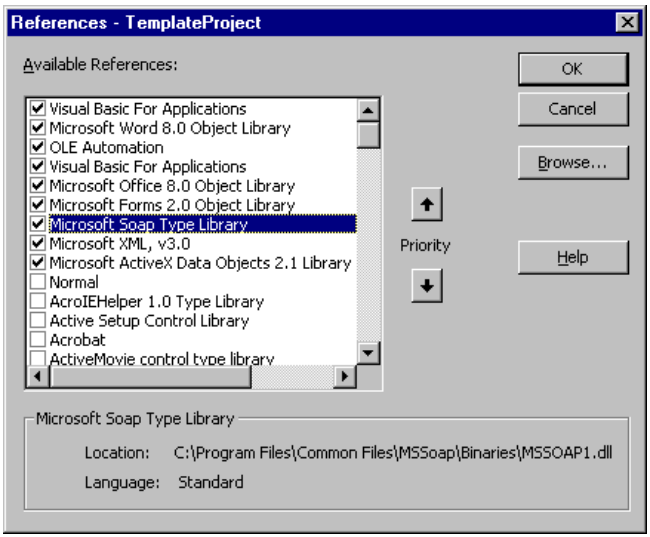


Fig. 8 – Integrating the libraries in the VisualBasic Editor

Architecture of the VBA Application

The following class modules are defined:

Table 74 – Class modules

Name	Description	WCM object type
`VipWebServiceClient`	Interface for using WCM WebServices. In addition to initialization by specifying the service URL, the user ID and password, this module provides all methods defined as functions in WCM WebServices.	
`ObjectId`	Encapsulates the OID of a WCM object Allows serialization and deserialization	`ObjectId`
`ObjectState`	Encapsulates the status of a WCM object Allows serialization and deserialization	`ObjectState`
`ObjectType`	Encapsulates the type of a WCM object Allows serialization and deserialization	`ObjectType`
`GenericFilter`	Generates the XML serialization of the WCM filter objects. As there is no inheritance in VBA, the implementation is quite complex.	`*Filter` (all WCM filters)
`User`	Serializes and deserializes the WCM `User` objects	`User`

Name	Description	WCM object type
`Value`	Serializes and deserializes the WCM `User` objects	`*Value (StringValue, DateValue, LongValue ...)`
`Version`	Serializes and deserializes the WCM `Version` objects	`Version`
`Locale`	Serializes and deserializes the Java `Locale` objects	`Locale`

Connection Setup (Module `VipWebServiceClient`)

The Web Service of the website “InternetSite” is available at the URL http://wcmserver.company.example/wcm/WebService/Port/InternetSite/edit. Please replace the individual parts of the URL by the values in your configuration.

The `SoapConnector` requires the following properties:

- `EndPointURL`
- `AuthUser` (user ID of the WCM user who is to access to data)
- `AuthPassword` (password of the WCM user)

```
'
' VipWebServiceClient
'
' SOAP Encoding Namespace URI
Private Const SOAP_ENCODING_NS_URI = _
                        "http://schemas.xmlsoap.org/soap/encoding/"
Private Const VIP_NAMESPACE_URI = "http://gaussvip.com/"

' instance variables
Private serializer As SoapSerializer
Private Connector  As SoapConnector
```

```
'   ------------------------------------------------------------
'   initialize_connection
'   ------------------------------------------------------------
Private Sub initialize_connection()
 If Connector Is Nothing Then
   Set Connector = New HttpConnector
   ' set "EndPointURL"
   Connector.Property("EndPointURL") = _
"http://wcmserver.company.example/wcm/WebService/Port/InternetSite/
edit"
   ' set "SoapAction", "AuthUser" and "AuthPassword"
   Connector.Property("SoapAction") = VIP_NAMESPACE_URI
   Connector.Property("AuthUser") = "admin"
   Connector.Property("AuthPassword") = "a"
 Else
   Connector.Reset
 End If
 ' initialise the SoapConnector Objekt and prepare the connection
 Connector.Connect

 ' Begin the SOAP Message
 Connector.BeginMessage

 ' create the necessary SoapSerializer
 Set serializer = New SoapSerializer

 ' connect the Serializer with the SOAP Connection
 serializer.Init Connector.InputStream

 ' Begin the SOAP Envelope
 serializer.startEnvelope

 ' add the following required XML Namespace definitions:
 serializer.SoapAttribute "xmlns:SOAP-ENC", , SOAP_ENCODING_NS_URI
 serializer.SoapAttribute "xmlns:xsi", , _
                       "http://www.w3.org/2001/XMLSchema-instance"
 serializer.SoapAttribute "xmlns:xsd", , _
                       "http://www.w3.org/2001/XMLSchema"
 serializer.SoapAttribute "xmlns:vip", , VIP_NAMESPACE_URI
' Begin the request body
 serializer.startBody
End Sub
```

Notes:

The SOAP toolkit does not support session management via cookies. Thus, WCM WebServices performs a separate login for each request. For all functions that change data (for example “Submit”), a valid WCM user license is required. WCM WebServices releases this license when the session times out or – at the latest – when the *Expiration interval* set for the Administration server expires after the last activity performed by the user.

For information on configuring the expiration interval, refer to the Livelink WCM Server Administrator Manual. To avoid license problems, you should use the `logout` method of WCM WebServices for explicit license release.

Calling a Method Without Return Value

After initialization of the `SoapSerializer`, the method can be inserted in the data flow with all its parameters. In the following sample code, this is shown on the basis of the `submit` method:

```
'   Submit an array of WCM objects (identified by their ObjectIDs) to QA
Public Function submit(ByRef oids() As ObjectID, _
                       ByVal pendingReleaseDate As Date, _
                       ByVal remark As String, _
                       ByVal emailI As emailInfo, _
                      ByVal dplWaitInfo As DeploymentWaitInfo) As Boolean
Dim success As Boolean
success = False
initialize_connection
' serialize request for submit
serializer.startElement "submit", VIP_NAMESPACE_URI

' Start element "oids"
'
serializer.startElement "oids"
    serializer.SoapAttribute "xsi:type", , "SOAP-ENC:Array"
    serializer.SoapAttribute "SOAP-ENC:arrayType", , _
             "vip:ObjectId[" & (UBound(oids) + 1 - LBound(oids)) & "]"
    Dim arrayOfOids As New Value
```

```
    Let arrayOfOids.ArrayValue = oids
    serializer.writeXML arrayOfOids.xml
serializer.endElement
' Start element "pendingReleaseDate"
'
serializer.startElement "pendingReleaseDate"
    Dim v As New Value
    Let v.dateValue = pendingReleaseDate

    ' pendingReleaseDate is only valid if it's in the future
    If pendingReleaseDate > Now Then
        serializer.SoapAttribute "xsi:type", , "xsd:dateTime"
        serializer.writeString v.simpleXml
    Else
        serializer.SoapAttribute "xsi:nil", , "true"
    End If
serializer.endElement
' Start element "remark"
'
serializer.startElement "remark"
    serializer.SoapAttribute "xsi:type", , "xsd:string"
    serializer.writeString remark
serializer.endElement
' Start element "emailInfo"
'
serializer.startElement "emailInfo"
    If Not emailI Is Nothing Then
        serializer.SoapAttribute "xsi:type", , "vip:EMailInfo"
        serializer.writeXML emailI.xml
    Else
        serializer.SoapAttribute "xsi:nil", , "true"
    End If
serializer.endElement
' Start element "dplWaitInfo"
'
serializer.startElement "dplWaitInfo"

    If Not dplWaitInfo Is Nothing Then
        serializer.SoapAttribute "xsi:type", , "vip:DeploymentWaitInfo"
        serializer.writeXML dplWaitInfo.xml
    Else
        serializer.SoapAttribute "xsi:nil", , "true"
    End If
serializer.endElement

serializer.endElement
' end of SOAP-Body
serializer.endBody
' end of SOAP-Envelope
```

```
serializer.endEnvelope
' end of SOAP message
Connector.EndMessage
' create a new SoapReader
Dim Reader As New SoapReader
' Load XML from OutputStream & display an error on failure
If Not Reader.Load(Connector.OutputStream) Then
    Dim reason As String
    reason = IIf(Reader.dom.parseError Is Nothing, _
              "parser: unknown reason", _
               Reader.dom.parseError.reason)
    MsgBox "submit failed:- " & reason, vbExclamation, "Error"
    Exit Function
End If
' if there is an application fault use a message box to report it
If Not Reader.Fault Is Nothing Then

    MsgBox "submit failed:- " & _
                Reader.faultstring.nodeTypedValue,_
                vbExclamation, "Error"
ElseIf Not Reader.Body Is Nothing Then
    submit = True
End If

End Function
```

This function shows very well how the parameters must be serialized to XML:

- the array with the OIDs is serialized via the `Value` class
- `EMailInfo` and `DeploymentWaitInfo` are classes of their own, each containing an `xml()` function.
- the remark field (of type `String`) is directly output here
- for the date field, the `Value` class is used

All invalid or missing parameters are represented by `xsi:nil="True"`. They can also be left out because Web Services identify the parameters by their names and replace missing parameters by the type-specific standard values (`False` for `Boolean`, `0` for `int`, as well as `long` and `null` for

complex types). The server response must be read carefully. The following error scenarios are possible:

- the server does not respond at all
- the response of the server does not contain XML
- the response of the server does not contain a SOAP envelope
- the response of the server does not contain a SOAP body
- the response of the server contains an application error

The example program tests these aspects. The return value of the function `submit(...)` indicates whether submitting to Quality Assurance has been performed successfully.

The focus of the next example is on the deserialization of the XML data supplied by the Web Service.

Calling a Method With Return Value

The evaluation of a filter expression is used as an example. As result, WCM WebServices returns an array of `ObjectData`. This array consists of `n` entries that reference their content via `href="#id..."`. In this example, XPath expressions are used to locate the respective elements in the DOM tree.

`ObjectData` itself is a complex data type, which references, among others, elements of the type `ObjectState`, `ObjectType`, `User`, `ArrayOfString`, and `ArrayOfObjectId`. These elements are also integrated via `href="#id..."`.

There are several ways of deserializing such a complex object by means of a simple programming language. Here, the following approach is used:

1. The SOAP response is loaded to a separate parser.

```
resultDom.loadXML(Reader.Body.xml)
```

2. The parser looks for the attribute `href` in the element `filterResult`.

```
result = resultDom.selectSingleNode("//filterResult/@href")
```

3. From the attribute, the value of the respective ID is extracted.

```
resultId = Mid(result.text, 2)
```

4. Thus, the array is located in the DOM.

```
List = resultDom.selectNodes("//*[@id='" & resultId & "']/*")
```

5. For each entry in the list, an `ObjectData` object is created.

```
itemId =_

Mid(List.Item(i).Attributes.getNamedItem("href").nodeValue, 2)
objectDataItem = resultDom.selectNodes("//*[@id='" &_
                                    itemId & "']/*")
results(i) = tmpObjectData.getObjectFromNodeList(_
                          objectDataItem, resultDom)
```

6. The individual `ObjectData` attributes are passed to the method `getObjectFromNodeList` as an `IXMLDOMSelection`.

These elements each contain the object type information (for example <title xsi:type="xsd:string"> PortalManager-ger_v01</title>). Depending on the type, the respective attribute is set in ObjectData. Complex types, such as ObjectState, each have their own getObjectFromNodeList(...) method.

The code of the filter method looks as follows:

```
'
' filter: returns an array of ObjectData containing the objects
' matching the specified filter and READ permissions for the
' authenticated user. The number of records returned is passed
' via the reference variable numRows.
'
Public Function filter(ByVal myFilter As GenericFilter, _
                       ByVal startOid As ObjectID, _
                       ByRef sortList() As Sort, _
                       ByVal startResult As Long, _
                       ByVal numberOfResults As Long, _
                       ByRef attributeKeys() As String, _
                       ByRef numRows As Integer) As Variant
initialize_connection
    numRows = 0
    ' serialize request for filter
    serializer.startElement "filter", VIP_NAMESPACE_URI

    ' Start element "filter"
    ' (xsi:type="vip:[EqualFilter|LessFilter ...])
    '
    serializer.startElement "filter"
     serializer.SoapAttribute "xsi:type", , "vip:" & myFilter.FilterType

     If Not myFilter Is Nothing Then
        serializer.writeXML myFilter.xml(serializer)
     End If
    serializer.endElement
    ' Start element "startOID"
    '
    serializer.startElement "startOID"
     serializer.SoapAttribute "xsi:type", , "vip:ObjectId"
     If Not startOid Is Nothing Then
            serializer.writeXML startOid.xml
     End If
    serializer.endElement
```

```
' Start element "sortList"
'
serializer.startElement "sortList"
 serializer.SoapAttribute "xsi:type", , "SOAP-ENC:Array"
 serializer.SoapAttribute "SOAP-ENC:arrayType", , _
         "vip:Sort[" & (UBound(sortList) - LBound(sortList)) & "]"

 Dim i As Integer

 For i = LBound(sortList) To UBound(sortList) - 1
     serializer.startElement "Item"
         serializer.SoapAttribute "xsi:type", , "vip:Sort"
         serializer.writeXML sortList(i).xml
     serializer.endElement
 Next i
serializer.endElement
' Start element "startResult"
'
serializer.startElement "startResult"
 serializer.SoapAttribute "xsi:type", , "xsd:int"
 serializer.writeString startResult
serializer.endElement
' Start element "numberOfResults"
'
serializer.startElement "numberOfResults"
 serializer.SoapAttribute "xsi:type", , "xsd:int"
 serializer.writeString numberOfResults
serializer.endElement
'
' start attributeKeys
serializer.startElement "attributeKeys"
 serializer.SoapAttribute "xsi:type", , "SOAP-ENC:Array"
 serializer.SoapAttribute "SOAP-ENC:arrayType", , _
         "xsd:string[" & (UBound(attributeKeys) -_
                          LBound(attributeKeys)) & "]"

 For i = LBound(attributeKeys) To UBound(attributeKeys) - 1
     serializer.startElement "Item"
         serializer.SoapAttribute "xsi:type", , "xsd:string"
         serializer.writeXML attributeKeys(i)
     serializer.endElement
 Next i
serializer.endElement

serializer.endElement
' close SOAP-Body
serializer.endBody
' close SOAP-Envelope
serializer.endEnvelope
```

```
' signal the end of the SOAP message being sent to the server
Connector.EndMessage

' create a new SoapReader
Dim Reader As New SoapReader

' load the XML from the Stream
If Not Reader.Load(Connector.OutputStream) Then
    Dim reason As String
    reason = IIf(Reader.dom.parseError Is Nothing, _
          "parser: unknown reason", _
           Reader.dom.parseError.reason)

    MsgBox "filter failed:- " & reason, vbExclamation, "Error"
    Exit Function
End If

' if there is a fault use a message box to report it
If Not Reader.Fault Is Nothing Then
    MsgBox "filter failed:- " & _
            Reader.faultstring.nodeTypedValue,_
            vbExclamation, "Error"
Else
    ' get the return Document
    Dim resultDom As DOMDocument30

    Set resultDom = New DOMDocument30
    resultDom.setProperty "SelectionLanguage", "XPath"
    resultDom.async = False
    resultDom.validateOnParse = False

    Dim success As Boolean
    ' set up the temporary results array
    Dim results() As objectData

    If Reader.Body Is Nothing Then
      success = False
    Else
      Let success = resultDom.loadXML(Reader.Body.xml)

      ' get the nodelist of array items from the response
      ' traverse to multiref with corresponding id
      Dim result As IXMLDOMNode
      Dim resultId As String

      ' look for corresponding attribute href=<xyy>
      Set result = resultDom.selectSingleNode(_
                                          "//filterResult/@href")
```

```
          If Not result Is Nothing Then
            resultId = result.text
            resultId = Mid(resultId, 2)

            Dim List, objectDataItemList As IXMLDOMNodeList

           Set List = resultDom.selectNodes("//*[@id='"&resultId&"']/*")

            If List.Length = 0 Then
                Exit Function
            End If

            ReDim results(List.Length - 1)
            numRows = List.Length

            ' for each array element
            For i = 0 To (List.Length - 1)
              ' Create Temporary "DocData"
              Dim tmpDocData As New objectData

              Dim itemId As String
              itemId =_
          Mid(List.Item(i).Attributes.getNamedItem("href").nodeValue, 2)

              Set objectDataItemList =_
                    resultDom.selectNodes("//*[@id='" & itemId & "']/*")
              ' Populate the array element
              Set results(i) =_
                   tmpDocData.getObjectFromNodeList(objectDataItemList,_
                                                          resultDom)
              Next

            End If
        End If
        ' set the return value
        filter = results
    End If
End Function
```

The filter method can, for example, be applied as follows:

```
Sub vipFilterPDF()
    Dim filteresult As Variant

    If VipWebService Is Nothing Then
```

```
        Set VipWebService = New VipWebServiceClient
    End If

    Dim pdfFilter As New GenericFilter

    Let pdfFilter.FilterType = "EqualFilter"
    Set pdfFilter.Key = New Key
    Let pdfFilter.Key.stringValue = "objectType"
    Set pdfFilter.Value = New Value
    Set pdfFilter.Value.objectType = New objectType
    Let pdfFilter.Value.objectType.name = "PDF"

    Dim startId As New objectID
    Let startId.id = "1"

    Dim dateSort(1) As New Sort
    Let dateSort(0).stringValue = "createdDate"

    Dim attributeKeys(4) As String
    Let attributeKeys(0) = "createdDate"
    Let attributeKeys(1) = "objectId"
    Let attributeKeys(2) = "objectState"
    Let attributeKeys(3) = "title"

    Dim numRows As Integer
    filterResult = VipWebService.filter(pdfFilter, startId, dateSort, _
                                        0, 5, attributeKeys, numRows)

End Sub
```

This example completes the code examples for Visual Basic. The complete source code for the modules listed in table “Class modules” on page 157 can be found in the directory **{WCM installation directory}\examples\webservices\vba**. Via *Import File*, you can integrate these classes in a Visual Basic project.

C# and ASP.NET

Visual Studio.NET considerably facilitates the development of browser-based user interfaces with ASP.NET or standard Windows user interfaces with Windows.Forms. This section explains how to use WCM WebServices in Visual Studio.

Preparations

Create a new C# project, for example with the name “VipWebService-Client”:

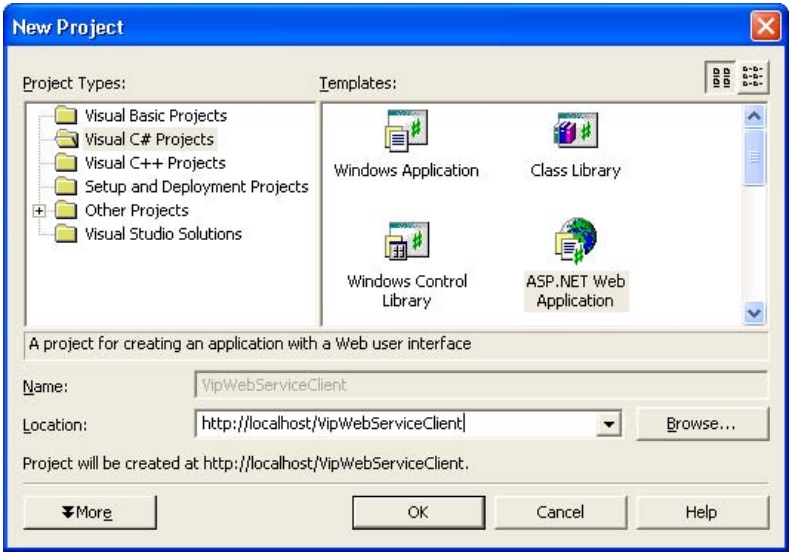


Fig. 9 – Creating a C# project

Visual Studio automatically creates the required modules for accessing WCM WebServices when you add the URL of the WSDL description as a web reference (menu *Projects* → *Add Web Reference*).

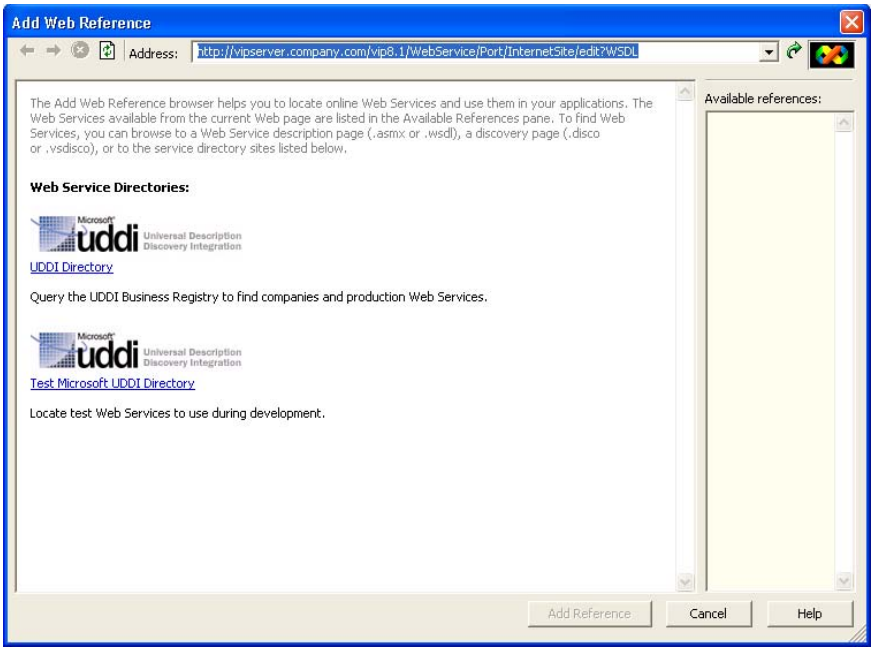

Add Web Reference
Address:
http://vipserver.company.com/vip8.1/WebService/Port/InternetSite/edit?WSDL
The Add Web Reference browser helps you to locate online Web Services and use them in your applications. The Web Services available from the current Web page are listed in the Available References pane. To find Web Services, you can browse to a Web Service description page (.asmx or .wsdl), a discovery page (.disco or .vsdisco), or to the service directory sites listed below.
Web Service Directories:
Microsoft uddi Universal Description Discovery Integration
UDDI Directory
Query the UDDI Business Registry to find companies and production Web Services.
Microsoft uddi Universal Description Discovery Integration
Test Microsoft UDDI Directory
Locate test Web Services to use during development.
Available references:
Add Reference
Cancel
Help

Fig. 10 – Adding the WSDL as a web reference

Visual Studio loads the WSDL into the left window pane and then prompts for the user name and password:

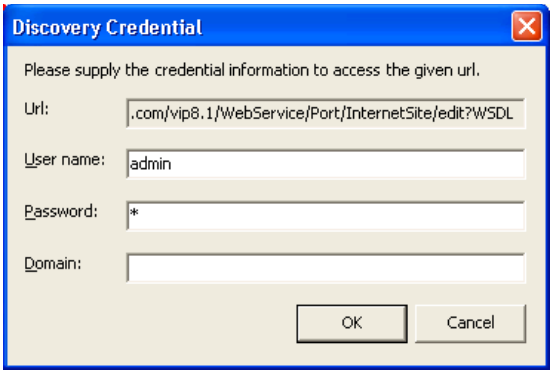


Fig. 11 – Entering user ID and password

After successful authorization, you can complete the *Add Web Reference* dialog by clicking *Add Reference*.

From the WSDL description, Visual Studio creates a WCM WebServices module in the *Namespace* `VipWebServiceClient.com.company.wcmserver`. Here you find a class for each data type required in WCM WebServices. The *Object Browser* offers a good overview of the automatically created classes:

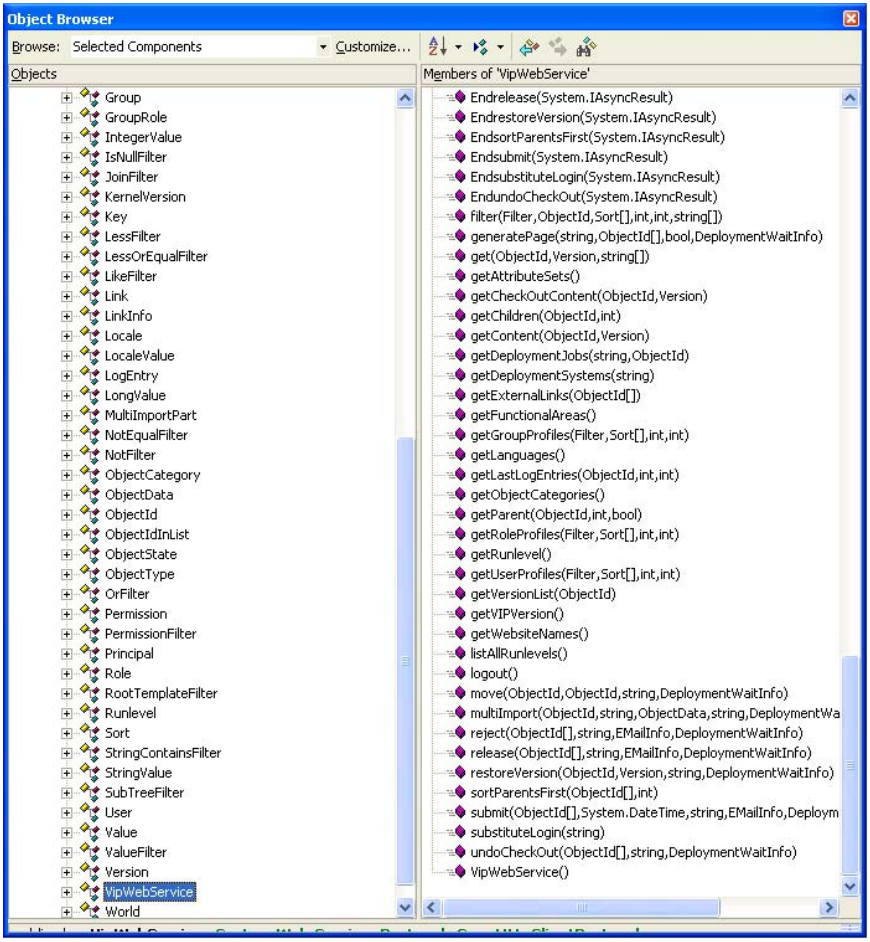


Fig. 12 – Automatically created classes in the namespace `VipWebServiceClient`

You can now start using WCM WebServices. The following example refers to an ASP.NET page (**default.aspx**) that lists all PDF files of the website "InternetSite" in a table. Each table row contains a box for selecting the row. All selected objects can be submitted/released by clicking a button.

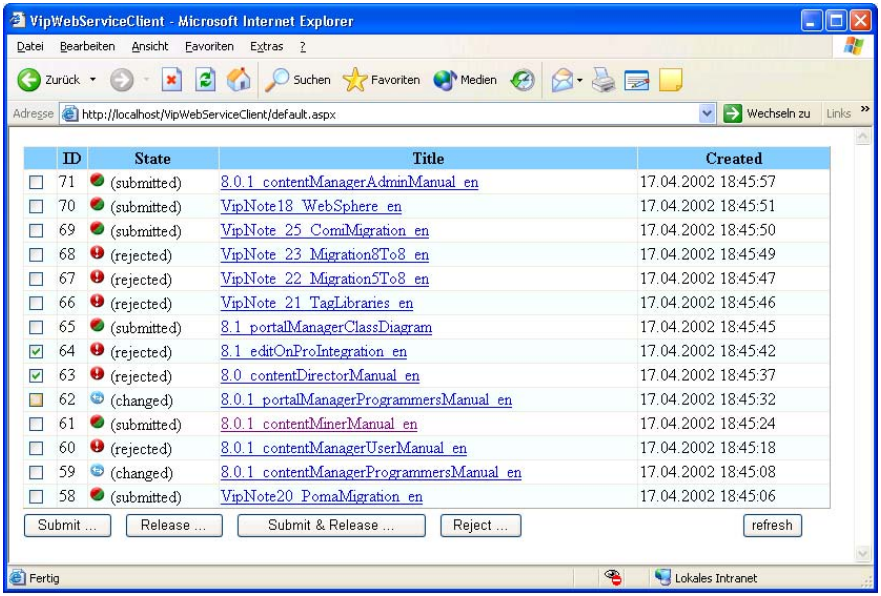

VipWebServiceClient - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://localhost/VipWebServiceClient/default.aspx
Wechseln zu Links
ID State Title Created
71 (submitted) 8.0.1_contentManagerAdminManual_en 17.04.2002 18:45:57
70 (submitted) VipNote18_WebSphere_en 17.04.2002 18:45:51
69 (submitted) VipNote_25_ComiMigration_en 17.04.2002 18:45:50
68 (rejected) VipNote_23_Migration8To8_en 17.04.2002 18:45:49
67 (rejected) VipNote_22_Migration5To8_en 17.04.2002 18:45:47
66 (rejected) VipNote_21_TagLibraries_en 17.04.2002 18:45:46
65 (submitted) 8.1_portalManagerClassDiagram 17.04.2002 18:45:45
64 (rejected) 8.1_editOnProIntegration_en 17.04.2002 18:45:42
63 (rejected) 8.0_contentDirectorManual_en 17.04.2002 18:45:37
62 (changed) 8.0.1_portalManagerProgrammersManual_en 17.04.2002 18:45:32
61 (submitted) 8.0.1_contentMinerManual_en 17.04.2002 18:45:24
60 (rejected) 8.0.1_contentManagerUserManual_en 17.04.2002 18:45:18
59 (changed) 8.0.1_contentManagerProgrammersManual_en 17.04.2002 18:45:08
58 (submitted) VipNote20_PomaMigration_en 17.04.2002 18:45:06
Submit ... Release ... Submit & Release ... Reject ... refresh
Fertig
Lokales Intranet

Fig. 13 – The sample application (list of all PDF files)

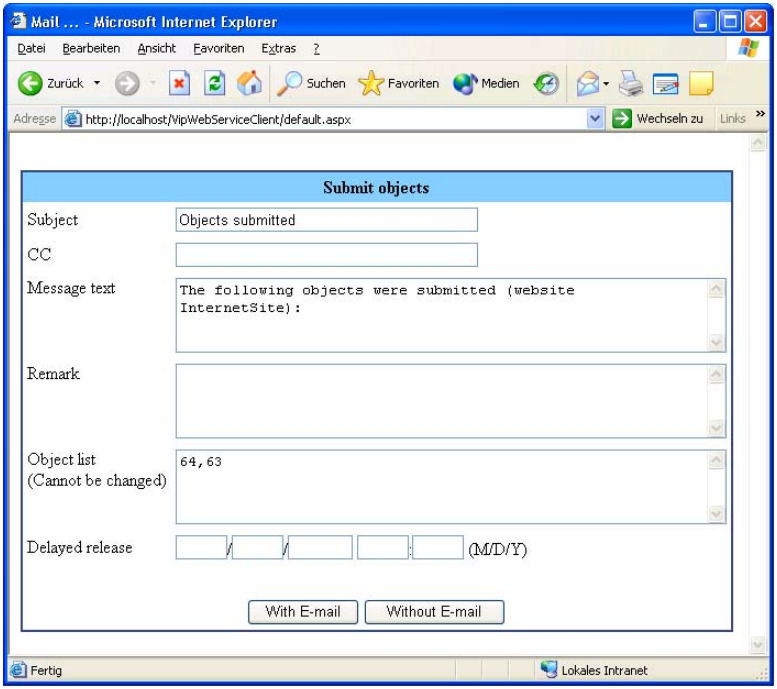


Fig. 14 – E-mail dialog of the example application

The example consists of the following files:

Table 75 – ASP.NET example

Name	Description	Elements used

default.aspx	ASP page: user interface
	Contains a `DataGrid` for displaying the object data
		`<asp:datagrid>`
		`<asp:label>`
		`<asp:TemplateColumn>`
		`<asp:CheckBox>`
		`<asp:HyperLink>`
		`<asp:button>`

default.aspx.cs	Application code behind the ASP page. This is where the data is loaded from Livelink WCM Server. Moreover, callbacks are implemented for the five buttons.	`CookieContainer`
		`CredentialCache`
		`NetworkCredential`
		`Server`
		`Session`
		`EqualFilter`
		`VipWebService`

mail.aspx	Simple e-mail input form (see figure “E-mail dialog of the example application” on page 175). Here you can enter additional e-mail recipients, specify the release date, and enter remarks for the object log.	`<asp:label>`
		`<asp:literal>`
		`<tr runat="server">`

mail.aspx.cs	The code behind the e-mail form. The line with the release date only becomes visible if the dialog box for submitting to QA is used. In the case of an error, a line with the error description is displayed at the bottom.	`VipWebService` `SoapException`

Connection Setup

The correct URL for connection setup is entered in the WSDL by WCM WebServices. It is then automatically inserted in the generated code by Visual Studio. What you have to do is to insert the code for authentication and the session tracking:

```
/// <summary>
/// Initializes the service and saves the service - reference within the
/// sesion (as 'service')
/// </summary>
private void intializeService()
{
      service = (VipWebService)Session["service"];
      if (service == null)
      {
         service = new VipWebService ();
         // allow reception of cookies
         service.CookieContainer = new CookieContainer();
         // create the necessary credentials
         // (using basic authentication)
         NetworkCredential credentials =
                  new NetworkCredential (userName, userPassword);

         CredentialCache credCache = new CredentialCache ();
         credCache.Add(new Uri(service.Url), "Basic", credentials);
         service.Credentials = credCache;

         Session["service"] = service;
         Session.Timeout = 60;
      }
}
```

Note: The .NET framework has a preset time limitation for synchronous Web Service calls. The default value is 3 minutes. With the exception of complex multi-imports, WCM WebServices calls are usually complete after a very short time. By means of the attribute `service.Timeout` (in milliseconds), you can set the time limitation to any other value. For example, `service.Timeout = 60000 * 10;` sets the time limitation to 10 minutes.

Structure of the Page

When the ASP page is loaded, the method Page_Load is called. Here the filter is executed via the Web Service:

```
private void Page_Load(object sender, System.EventArgs e)
{
      // Put user code to initialize the page here
      // initialize the service ...
      intializeService();

      if (! IsPostBack)
      {
         // load the data into the DataGrid
         tableOfObjects.DataSource = filterPDF();
         tableOfObjects.DataBind();
      }
}
```

The actual filtering is swapped out to the following method:

```
/// <summary>
/// Retrieves all PDF files via a WCM Web Service call
/// </summary>
/// <returns></returns>
private ObjectData[] filterPDF()
{
      EqualFilter filter = new EqualFilter ();
      filter.key   = new Key();
      filter.value = new ObjectType();

      filter.key.stringValue = "objectType";
      ((ObjectType)filter.value).name = "PDF";

      ObjectId startId = new ObjectId ();
      startId.id = "1";

      Sort[] sortList = new Sort[]{ new Sort() };
      sortList[0].stringValue = "createdDate";
      sortList[0].descending  = true;

      String[] attributeKeys = new String []{   "objectId",
                                        "objectState",
                                        "title",
```

```
                                        "createdDate",
                                        "createdBy",
                                        "modifiedDate",
                                        "modifiedBy",
                                        "URL"};

      return service.filter (filter, startId, sortList,
                             0, 100, attributeKeys);
}
```

This code again shows how easy it is to use the methods of WCM WebServices. Due to the “CodeCompletion” technique in Visual Studio, you immediately get an overview of the available methods, data types, and attributes.

Structure of the Table

For displaying the filter results, a `DataGrid` control is used. It offers sufficient flexibility for displaying the data.

So-called `TemmplateColumns` are responsible for the presentation of the data. The `ItemTemplate` provides full access to the content of the current table row (in this case an `ObjectData` object). For the representation of the results, the property `AutoGenerateColumns` of the `DataGrid` must be set to `False`.

The `ItemTemplates` looks as follows:

```
<asp:TemplateColumn HeaderText="ID">
      <ItemTemplate>
         <asp:Label id="C_OID" runat="server"
        Text="<%#((ObjectData)Container.DataItem).objectId.id%>">
         </asp:Label>
      </ItemTemplate>
</asp:TemplateColumn>
```

You can also use several elements in one table cell: in this case, for example, the icon for the object status followed by the description:

```
<asp:TemplateColumn HeaderText="State">
  <ItemTemplate>
    <IMG src="<%#((ObjectData)Container.DataItem).objectState.imageURL
              %>">
        (<%#((ObjectData)Container.DataItem).objectState.description %>)
  </ItemTemplate>
</asp:TemplateColumn>
```

The first column of the table contains a `CheckBox` which is implemented as a server-side control element here. Each click on a `CheckBox` is directly transmitted to the server via the `PostBack` function.

```
<asp:TemplateColumn>
      <ItemTemplate>
         <asp:CheckBox id="cb1" runat="server" AutoPostBack="True">
         </asp:CheckBox>
      </ItemTemplate>
</asp:TemplateColumn>
```

On the server, the `ObjectId` is extracted from all rows selected by means of the `CheckBox` and passed to the e-mail dialog box for further processing (**mail.aspx**).

Structure of the E-Mail Dialog Box

For the different staging actions (submit, release, direct release, reject), the e-mail dialog box only differs in the following respects:

- the texts are different
- entering the desired release date is possible when submitting or in the case of direct release

For this reason, the respective table row is realized as a server-side control element (`id="trDelayed"`) and is accordingly displayed or hidden in the method `Page_Load()`.

```
String operation = Request.Params["operation"];
trDelayed.Visible = (   operation.Equals("submit")
                     || operation.Equals("directRelease"));
```

Performing the Action

If one of the two buttons in the e-mail dialog box is clicked, the desired action is performed. For this purpose, the input data from the dialog box is inserted in an `EmailInfo` object and supplied as parameters to the desired method:

```
private void execute(Boolean withMail)
{
      VipWebService service = (VipWebService) this.Session["service"];
      String operation = Request.Params["operation"];

      if (service != null)
      {
         String[] oids =
                     textAreaObjectList.Value.Split(",".ToCharArray());

         ObjectId[] oidsToSubmit = new ObjectId[oids.Length];
         for (int i = 0; i < oids.Length; i++)
         {
            oidsToSubmit[i] = new ObjectId ();
            oidsToSubmit[i].id = oids[i];
         }
         DateTime pendingReleaseDate;

         pendingReleaseDate = new DateTime(1,1,1);

         EMailInfo eMailInfo = null;
         if (withMail)
         {
            eMailInfo         = new EMailInfo ();
            eMailInfo.subject = textAreaMessageText.Value;
            eMailInfo.cc      = textAreaCC.Value;
         }
         DeploymentWaitInfo dplWaitInfo = null;
         //
         try
         {
```

```
            switch(operation)
            {
               case "submit":
                  service.submit (oidsToSubmit, pendingReleaseDate,
                                  textAreaRemark.Value, eMailInfo,
                                   dplWaitInfo);
                  break;

               case "directRelease":
                  service.directRelease (oidsToSubmit,
                                         pendingReleaseDate,
                                         textAreaRemark.Value,
                                         eMailInfo,
                                   dplWaitInfo);
                  break;

               case "release":
                  service.release (oidsToSubmit,
                                  textAreaRemark.Value, eMailInfo,
                                   dplWaitInfo);
                  break;

               case "reject":
                  service.reject (oidsToSubmit,
                                  textAreaRemark.Value, eMailInfo,
                                   dplWaitInfo);
                  break;
            }
            // ... and return to the calling page
            Server.Transfer (nextURL);
         }
         catch (Exception exc)
         {
            Exception e1 = exc;
            rowError.Visible = true;
            literalError.Text = exc.Message;
         }
      }
}
```

The desired action is executed in a try/catch block. This is where all exceptions are caught.

Error Handling

It is in the nature of distributed applications that there are more possible error sources. This means that you as the application developer have to accurately evaluate the error conditions and present them in a form that the end user can understand.

The possible error sources include:

- connection problems to WCM WebServices (WCM server not available)
- authentication problems (wrong user or wrong password)
- expiration of the time limitation for the call (timeout too short)
- SOAP error (client or server send SOAP envelopes that contain errors or cannot be interpreted correctly)
- error messages of the WCM API (operation not allowed, system error, such as an error when accessing the database)

All the above errors are described by a `Message` in the `Exception` thrown. This message is displayed in the mail dialog box shown here.

The error messages of WCM WebServices are automatically output in the language of the user. In addition to the short error text addressing the end user, the error messages of Livelink WCM Server contain more detailed information, which are, in particular, important for system administrators and the Technical Support of Gauss Interprise AG. As in the Content client, this information can be displayed in detail.

The error code is available via the attribute `Code` (of type `System.Xml.XmlQualifiedName`) of the `SoapException`. For a list of server error codes, refer to section “Exceptions of the Content Server” on page 47.

All detailed information is available via the attribute `Detail` (of type `System.Xml.XmlNode`) of the `SoapException`.

The following code excerpt contains the SOAP message with the error code:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:tns="http://gaussvip.com/"
                xmlns:types="http://gaussvip.com/encodedTypes"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>

    <faultcode xmlns:ns1="http://gaussvip.com/">
                 ns1:Server.ActionNotPermittedException
    </faultcode>
    <faultstring>
         0582: No direct release is permitted for object &apos;68&apos;.
    </faultstring>
    <faultactor>Vip-Content-Manager-API
    </faultactor>
    <detail>

     <tns:message sequenceNo="0">
         0303: The master server was unable to perform the request.
     </tns:message>
     <tns:message sequenceNo="1">
         0738: Direct release of object &apos;68&apos; failed.
     </tns:message>
     <tns:message sequenceNo="2">
         0582: No direct release is permitted for object &apos;68&apos;.
     </tns:message>
     <tns:wrappedExceptionName>de.gauss.vip.api.VipApiException
     </tns:wrappedExceptionName>

     <tns:wrappedExceptionMessage>
         0303: The master server was unable to perform the request.
     </tns:wrappedExceptionMessage>
    </detail>
   </soap:Fault>
  </soap:Body>
 </soap:Envelope>
```

Thus, a detail display could show the `<tns:message>` elements. The following rule applies: the higher is the sequence number (attribute `sequenceNo`), the more specific is the message.

The source code may look as follows:

```
catch (SoapException soapExc)
{
      rowError.Visible = true;
      literalError.Text = soapExc.Message
                     + "<hr/>Code: " + soapExc.Code.Name
                     + "<hr/>Details<br/>";

      XmlNodeList details = soapExc.Detail.ChildNodes;

      for (int i = 0; i < details.Count; i++)
      {
         XmlNode detail = details[i];
         if (detail.LocalName.Equals ("message"))
         {
            literalError.Text += "<br/>" + detail.FirstChild.Value;
         }
      }
      literalError.Text += "<hr/>";
}
catch (Exception exc)
{
      rowError.Visible = true;
      literalError.Text = exc.Message;
}
```

This completes the code examples for C# and ASP.NET. The complete source code for the modules listed in table “ASP.NET example” on page 176 can be found in the directory **{WCM installation directory}\examples\webservices\asp.net**. Via *Add existing item*, you can integrate these files in a Visual Studio project.

Caveats

The following has to be considered when using WCM WebServices in the .NET environment: there is a name conflict between the WCM data type `Version` and the built-in .NET data type `Version` in the namespace `System`. For this reason, the WCM data type `Version` must always be fully qualified with namespace.

Example:

```
VipWebServiceClient.com.company.wcmserver.Version v =
             new VipWebServiceClient.com.company.wcmserver.Version();
 ...
```

Glossary

Access control list – For each WCM object, users, groups, roles, and group-roles that have access to this object can be specified. The individual access rights are specified separately for each principal authorized to access the object. Also called ACL.

ACL – Access Control List

Action – Step required to manage website objects. It generally results in a change in status of the object in question.

Asynchronous action – An asynchronous action returns almost immediately after it has been called. It does not wait for processing of the associated tasks to finish. See also *Synchronous action*.

Check in – Staging action of Livelink WCM Server. An object that has been checked out and edited is returned to the WCM system by the action "Check in". This makes the changes visible in the Edit view. The object is combined with the template again and is available for further editing.

Check out – Staging action of Livelink WCM Server. The content of a WCM object can only be edited after the object has been checked out. A checked-out object is locked for access by other users. The changes to the object do not become available in the Edit view until the object has been checked in.

Data storage view – The data storage view of a server refers to the aspects of the WCM objects that are currently available (Edit view, QA view, Production view). The data storage view available on a server is determined in the routing settings of the website.

Deployment – Deployment is the distribution of data. The deployment of Livelink WCM Server performs two main tasks: first, generating pages from the WCM objects stored in the database and distributing the generated files to the appropriate directories; second, notifying the WCM servers of changes in the WCM system.

Deployment system – The deployment systems generate pages from the WCM objects and distribute the generated files to the appropriate directories. From there, the files become visible for the users via an HTTP server. Deployment systems may be of various types and categories.

Deployment system category – Depending on the way of processing deployment jobs, deployment systems are assigned to various categories: *Standard deployment systems* automatically generate a new page every time a WCM object is changed. The generated pages are stored in the file system. *Dynamic deployment systems* generate the pages on the basis of user-defined settings and only when the page is requested via the HTTP server. The generated files are stored in a flat file structure. By means of *Search engine deployment systems*, you can prepare your website data for use with a search engine. *WebDAV deployment systems* are required for the use of WebDAV clients. *InSite Editing deployment systems* provide the basis for editing and adding content directly in the website – without the Content client.

Deployment system types – On the basis of the staging concept of Livelink WCM Server, a distinction is made between deployment systems of type “Edit”, “QA”, and “Production”. Different views of the website data are generated, depending on the type.

Edit view – In the Edit view of Livelink WCM Server, the objects of a website are created and edited. Here the most current status of the objects is visible.

Navigation topology – Structure of WCM objects within a website, organized according to topics (topic structure)

Object category – Assignment of a WCM object to a specific category. Due to this assignment, the WCM object has a set of additional special attributes (metadata).

Object data – Components of a WCM object: content, metadata, and log

Object type – The specific kind of object, e.g. “HTML page”, “HTML template”, “Topic”. Various properties of the WCM object result from the

object type. The object type is defined when the object is created. There are only a few cases in which it may subsequently be changed. Object types can be edited in the Admin client or the Content client.

Production view – The Production view of Livelink WCM Server makes the released pages of a website available to the user. By means of a web server, these pages can be accessed in the Internet, intranet, or extranet.

QA view – The QA view of Livelink WCM Server is used for quality assurance of the objects and thus of the website content. This view thus performs the control function between editing in the Edit view and publication in the Production view.

Release – Staging action of Livelink WCM Server. Quality Assurance checks whether the content and the form of a submitted object meet the quality standards of the company. If so, the object is released. The release transmits the quality-assured version of the object to the Production view, and thus makes it available to the end user in the published website.

Session – Unit managed by the JSP engine so that logically related actions (in terms of resources) can be combined

SOAP – Simple Object Access Protocol. Standard that makes platform-independent access to Web Services possible. It defines the data transmission between the provider and the requester of the Web Services. The exchange format used is XML.

Status – The processing state of a WCM object. Changes to the state are caused by the corresponding actions on the WCM object.

Submit – Staging action of Livelink WCM Server. before a newly created or changed object can be published, it must be submitted to Quality Assurance for checking. This makes the changes to the object visible in the QA view.

Synchronous action – A synchronous action does not return until all the tasks involved are completed. See also *Asynchronous action*.

Template topology – Structure of WCM objects within a website, organized according to templates (template structure)

Topology – A hierarchic organization of WCM objects according to specific criteria. In addition to the *Navigation topology* (organization according to topics with their respective subordinate objects) there is also a *Template topology*.

Transaction – A transaction can be used to group together several actions so that none of the changes involved actually becomes effective until the transaction is finalized (committed).

UDDI – Universal Description, Discovery and Integration. This XML-based standard specifies how detailed information on Web Services and their providers can be saved in a uniform way in directories.

VIPP – VIP Protocol. A proprietary protocol for exchanging data between the components of a WCM system. VIPP can be tunneled in HTTP for communication in WANs or over the Internet.

Web Services – XML-based standard for interfaces. Web Services allow the communication between programs written in different programming languages and running on different platforms via the standardized protocol SOAP. Another great advantage of Web Services is the use of standard web formats and protocols: XML, HTTP, and TCP/IP.

Workflow definition – Description of a workflow. A workflow definition consists of starting points, end points, activities, and transitions. Workflow definitions are saved in XPDL format and can be edited graphically by means of the Content Workflow Modeler.

WSDL – Web Services Description Language. Description language for Web Services on the basis of XML. For an application to use Web Services, an exact “language regulation”, i.e. description, is required. The description must provide detailed information about the protocol used, the address and the port number, the available procedures and functions as well as input and output formats. The provider of a Web Service makes this information available in the form of a WSDL file.

XML – Extensible Markup Language. A standard for defining the formal structure of documents.

XSL – Extensible Stylesheet Language. A standard for commands used for converting XML documents to other formats. XSL is often used for converting XML to HTML.

Index

A

access control list 86
  change 87
access rights 84
ACL
  change 87
Acl 86
AclEntry 87
Activity 91
addRemark 116
ASP.NET
  example 170
  ItemTemplate 179
  Page_Load 178
assignWorkflow 104
assignWorkflowAsync 105
attribute sets
  request 61
attributes of a WCM object
  request 128
authentication 172
  C# 177
  SOAP toolkit 158

C

C#
  cookies 177
  example 170
  filter 178
change 117
  ACL 87
  metadata 117
changePassword 68
check in 95
check out 96
  undo 103
checkIn 95
checkOut 96
checkReferencesForDelete 118
checkReferencesForRelease 118
checkReferencesForSubmit 119
child objects of a WCM object
  request 130
compound object
  convert 120
connection setup
  C# 177
content of a WCM object
  get 131
convertContent 120
cookies
  C# 177
copy 121
create 122
  WCM objects 122

D

data type
  Acl 86
  AclEntry 87
  DeploymentWaitInfo 93
  EMailInfo 94
  MultiImportPart 88
  ObjectData 74
  ObjectId 73
  ObjectState 81
  ObjectType 80
  Permission 84
  Workflow 90

data types
in general 72
delete 124
finally 126
page from file system 125
deployment
request jobs 132
deployment content of a WCM object
request 129
deployment systems 93
request 63
DeploymentWaitInfo 93
depublishPage 125
destroy 126
direct release 98
directRelease 98, 181
DOM 163

E

edit
ACL 87
e-mail
data 94
EMailInfo 94, 181
error handling 163
C# 183
SOAP example 183
example
ASP.NET 170
C# 170
e-mail dialog 175
VisualBasic 150
external references of a WCM object
request 132

F

file type
Activity 91
Transition 92
WorkflowNavigationInfo 92
filter 145, 163
filter WCM objects 145, 165, 178
forward 106
functional areas
request 61

G

generate
page 127
generatePage 127
get 128
getAssignedJobs 107
getAssignedWorkflow 108
getAssignedWorkflowActivities 109
getAssignedWorkflows 109
getAttributeSets 61
getAvailableWorkflows 110
getCheckOutContent 129
getChildren 130
getContent 131
getDeploymentJobs 132
getDeploymentSystems 63
getExternalLinks 132
getFunctionalAreas 61
getGroupProfiles 55
getLanguages 66
getLastLogEntries 133
getObjectCategories 62
getObjectTypes 62
getParent 134
getRoleProfiles 56
getRunlevel 66
getUserProfiles 57
getVersionList 135
getVIPVersion 67

getWebsiteNames 62
getWorkflow 110
group data
request 55

I

importing linked objects 137
isForwardable 111
isRemovable 111
ItemTemplate 179

L

languages
request 66
listAllRunlevels 66
log entries of a WCM object
request 133
logging out 68
logout 68

M

metadata 74
methods
addRemark 116
assignWorkflow 104
assignWorkflowAsync 105
change 117
changePassword 68
checkIn 95
checkOut 96
checkReferencesForDelete 118
checkReferencesForRelease 118
checkReferencesForSubmit 119
convertContent 120
copy 121
create 122
delete 124
depublishPage 125
destroy 126
directRelease 98, 181
filter 145
forward 106
generatePage 127
get 128
getAssignedJobs 107
getAssignedWorkflow 108
getAssignedWorkflowActivities 109
getAssignedWorkflows 109
getAttributeSets 61
getAvailableWorkflows 110
getCheckOutContent 129
getChildren 130
getContent 131
getDeploymentJobs 132
getDeploymentSystems 63
getExternalLinks 132
getFunctionalAreas 61
getGroupProfiles 55
getLanguages 66
getLastLogEntries 133
getObjectCategories 62
getObjectTypes 62
getParent 134
getRoleProfiles 56
getRunlevel 66
getUserProfiles 57
getVersionList 135
getVIPVersion 67
getWebsiteNames 62
getWorkflow 110
isForwardable 111
isRemovable 111
listAllRunlevels 66
logout 68
move 136
multiImport 137
reject 99, 181
release 100, 181
removeWorkflow 112
removeWorkflowAsync 113
restoreVersion 139

sortParentsFirst 140
submit 101, 181
submitImmediately 102
substituteLogin 68
undoCheckout 103
move 136
multiImport 137
MultiImport information 88
MultiImportPart 88

O

object categories
request 62
object serialization 152
object type 80
request 62
ObjectData 74
ObjectId 73
ObjectState 81
ObjectType 80
OID 73
old version of a WCM object
retrieve 139

P

parent objects of a WCM object
request 134
password
change 68
Permission 84

R

reject 99, 181
release 100, 181
remark
add to log 116
removeWorkflow 112
removeWorkflowAsync 113
restoreVersion 139
role data
request 56
run level
request all 66
request current 66

S

search
WCM objects 145
session management
C# 177
SOAP toolkit 160
session timeout 177
SOAP
body 150
envelope 150
example 150
SOAP toolkit 156
authentication 158
connection setup 158
sortParentsFirst 140
status of a WCM object 81
submit 101, 102, 181
submitImmediately 102
substitute
log in 68
substituteLogin 68

T

time limitation 177
timeout 177
Transition 92

U

undoCheckout 103
user data
request 57

V

version 186
    get back 139
    retrieve 67
version list of a WCM object
    request 135
VisualBasic 150
VisualStudio 170
    create project 170
    web reference 170

W

WCM object
    attributes 74
Web Services
    in general 19
website names
    request 62
Workflow 90
workflow
    assign 104
    remove 112
WorkflowNavigationInfo 92
WSDL
    use in VisualStudio 170

X

XML
    serialization 152
XML serialization
    example 162
    href 163
    mulitRef 163
XPath 163